.::shell scripts::.

d u n e t n a . k e r n e l p a n i c

Índex

0. intro

Aquests manual, tutorial o com li vulgueu dir és el resultat del curs realitzat per kernelpanic (hacklab de Barcelona) de programació de shell scripts.

No pretén ser un gran manual sinó una ajuda a aquelles persones que, ja sabent programar i tenint uns coneixements genèrics de UNIX, volen usar el llenguatge que ofereix la shell per fer scripts que els facilitin la vida ;-)

Dediquem aquests apunts al CSO Les Naus, desallotjat el 9 de desembre del 2003. Ens trauran els espais però seguirem lluitant, construint i compartint coneixements i il·lusions.

NOTA: Podeu trobar els exemples d'scripts que tenen el seu nom entre parèntesi a la web http://kernelpanic.hacklabs.org. Per a qualsevol dubte o suggerència podeu enviar un mail a info@kernelpanic.hacklabs.org.

1. conceptes bàsics

1.1. variables

La shell ens permet definir variables on emmagatzemar dades. Tenim dues zones de memòria on podem definir les nostres variables: l'àrea local i l'entorn.

Variables locals: només són visibles per la shell on estem treballant, no són visibles per cap subshell, és a dir, no són visibles per cap subprocés de la shell.

Variables d'entorn: són visibles tant per la shell on estem com per qualsevol subshell que obrim (o per qualsevol subprocés de la shell)

NOTA: una variable declarada en un procés fill no serà visible pel seu procés pare (encara que sigui d'entorn)

COMANDES	
set	Veure totes les variables definides
env	Veure totes les variables d'entorn definides
nom_var=valor_var	Definir variable i assignar-li valor
export *nom_var=valor_var*	Definir variable entorn i assignar-li valor
export *nom_var*	Convertir variable local a variable d'entorn
unset *nom_var*	Alliberar una variable

::exemple::

```
$ varmeva=3              declaro una variable local
$ echo $varmeva          miro el seu valor
3
$ bash                   entro en una subshell
$ echo $varmeva          no té valor perquè és local
$ exit                   tornem a la shell principal
$ export varmeva         converteixo la var. local a var. d'entorn
$ echo $varmeva          a la shell principal segueixo veient el seu
3                        valor 3
$ bash                   entro en una subshell
$ echo $varmeva          ara sí que veiem el seu valor!!! (és
3                        d'entorn)
$ exit
```

Variables predefinides: Disposem també d'una sèrie de variables ja definides que ens poden ser de gran ajuda per tal d'obtenir o desar informació genèrica.

VARIABLES PREDEFINIDES	
HOME	Directori de treball actual
PATH	Llocs on podem accedir directament sense escriure el camí
PS1	Prompt primari

VARIABLES PREDEFINIDES	
PS2	Prompt secundari
BASH	Camí del programa bash
BASH_VERSION	Versió de la bash actual
COLUMNS	Número de columnes a la pantalla
GROUPS	Identificador del grup principal de l'usuari
HISTCMD	Índex a l'històric de la comanda actual
HISTFILE	Fitxer on es guarda l'històric
HISTFILESIZE	Mida del fitxer històric
HOSTNAME	Nom de la màquina
LANG	Idioma per defecte, si no s'ha especificat a cap LC_
LC_ALL	Idioma
PID	Identificador del procés actual
PWD	Camí on estem situats
RANDOM	Número aleatori
SECONDS	Segons que porta encesa la màquina
UID	Identificador de l'usuari actual

::exemple::

$ echo "El meu directori de treball és $HOME i el meu id de grup és $GROUPS"
El meu directori de treball es /home/kp i el meu id de grup és 1000

1.2. substitució de variables

Substitució de variables: És la tècnica que farem servir per fer referència al valor contingut en una variable.

SUBSTITUCIÓ DE VARIABLES	
$nom_var	Fer referència al valor de la variable *nom_var*
${nom_var}	Fer referència al valor de la variable nom_var. Les claus ens delimiten el nom de la variable.

::exemples::

1::: $ saluda="hola" defineixo variable saluda amb valor "hola"
$ nom="anna" defineixo variable nom amb valor "anna"
$ echo saluda nom mostro per pantalla els valors de les variables...
saluda nom està malament: he posat els noms de les vars, no el seu valor
$ echo $saluda $nom ara està bé!
hola anna

2::: $ masc="gat" defineixo variable masc amb valor "gat"
$ echo masculí:$masc femení:$masca mostro el $masc i $masc seguit d'una a
masculí:gat femení: m'interpreta masca com una var (buida)

```
$ masculí:$masc femení:${masc}a     poso {} per diferenciar el
                                    nom de la variable
masculí:gat femení:gata     això és el que volia!!!
```

1.3. substitució de comandes

Substitució de comandes: És la tècnica que farem servir per substituir el nom d'una comanda per la sortida d'aquesta.

SUBSTITUCIÓ DE COMANDES	
`*comanda*` **$(*comanda*)**	Substituir el nom de la comanda per la seva sortida

::exemples::

1:::
```
# whoami                executem una comanda
root                    que ens diu com a què estàs connectat/da
# echo Sóc el/la whoami  si volem mostrar per pantalla i posem
                         la comanda...
Sóc el/la whoami        ...ens surt literalment el que li posem
# echo Sóc el/la `whoami`  substituïm whoami pel valor que retorna
                           i...
Sóc el/la root          ...obtenim el que volíem!!!
# echo Sóc el/la $(whoami) una altra manera de fer-ho
Sóc el/la root
```

2:::
```
$ date +%F; date +%D        executem date amb formats %F i %D
2004-10-21
10/21/04
$ aaaammdd="%F"             donem com a valors aquests formats a dues
                            variables
$ mmddaa="%D"
$ echo "Data (aaaa-mm-dd):`date +$aaaammdd`"      substituim
Data (aaaa-mm-dd):2004-10-21                      comandes i
$ echo "Data (mm/dd/aa):`date +$mmddaa`"          vars..
Data (mm/dd/aa):10/21/04
```

3::: petit script que ens crea una còpia de seguretat del nostre directori de treball:
```
CS=/var/backup-`date +%Y%m%d`.tgz  variable amb el nom de la
                                   còpia de seguretat
                                   (ex: /var/backup-20040601)
tar -czf $CS /home/nomusu  comprimim i empaquetem el directori de
                           treball
```

1.4. caràcters especials

Caràcters especials: Hi ha caràcters que per a la shell tenen un significat especial. Existeixen diferents tècniques per tal que la shell ignori aquest significat o el tingui en compte.

COMANDES	
\	anul·la el significat especial del caràcter que va darrera
' '	anul·la el significat especial de tots els caràcter que estiguin dins les cometes
“ “	anul·la el significat especial de tots els caràcters excepte: $ \ `` “”

::exemples::

1:::
```
$ echo “El “silenci”” volem mostrar silenci entre “”...
El silenci ens interpreta les “ de silenci com a caràcter especial!
$ echo El \”silenci\” les hem d'”escapar”
El “silenci”
```

2:::
```
# echo 'Sóc el/la $LOGNAME i estic a $PWD' amb '' ...
Sóc el/la $LOGNAME i estic a $PWD ...no interpreta $
# echo “Sóc el/la $LOGNAME i estic a $PWD” amb “” ...
Sóc el/la root i estic a /root ...sí que els interpreta!
# echo “Sóc el/la $LOGNAME i \$PWD: $PWD” barrejant “” i \ ...
Sóc el/la root i i $PWD: /root ...podem interpretar o no segons ens convingui
```

1.5. redireccionament e/s

stdin*, *stdout* i *stderr*:** Hi ha comandes que accepten les dades d'entrada pel que anomenem entrada estàndard (stdin***, amb descriptor de fitxer 0), que és el teclat. També hi ha comandes que ens donen la seva sortida pel que anomenem sortida estàndard (***stdout***, amb descriptor de fitxer 1), que és la pantalla. Per últim, tots els errors que pugui produir una comanda es dirigeixen a la sortida d'errors (***stderr***, amb descriptor de fitxer 2)

Redireccionament d'E/S: El redireccionament d'E/S ens permetrà que prenem les dades d'entrada, sortida i error i les redireccionem cap a un altre fitxer diferents als donats per defecte (*stdin*, *stdout*, *stderr*)

REDIRECCIONAMENT E/S	
<	redireccionament d'*stdin*
>	redireccionament d'*stdout* si el fitxer no existeix el crea si el fitxer existeix es carrega el contingut
>>	redireccionament d'*stdout* si el fitxer no existeix el crea si el fitxer existeix afegeix a continuació
2>	redireccionament d'*stderr* si el fitxer no existeix el crea si el fitxer existeix es carrega el contingut
2>>	redireccionament d'*stderr* si el fitxer no existeix el crea si el fitxer existeix afegeix a continuació
1>&2	redireccionar *stdout* a *stderr*
2>&1	redireccionar *stderr* a *stdout*
>&	redireccionar *stdout* i *stderr* a un fitxer

::exemples::

1::: `$ mail root -s "hack your mind!" < mailpr` mailpr seria un fitxer amb el missatge

2::: `$ cat fit1 fit2` fit1 existeix, fit2 no existeix

```
això és el contingut del fitxer 1          stdout
cat: fit2: No existe el fichero o el directorio    stderr
```

```
$ cat fit1 fit2 >> copiafit 2>> error.log
```

afegim fit1 al fitxer copiafit i l'error anirà a error.log

3::: `$ cat fit1 fit2 2> fit3 1>&2` (fit1 existeix, fit2 no existeix)

totes les sortides estan a fit3 perquè els errors van a fit3 i redirigim *stdout* a *stderr*

`$ cat fit1 fit2 > fit3 2>&1` (fit1 existeix, fit2 no existeix)

totes les sortides estan a fit3 perquè *stdout* va a fit3 i redirigim *stderr* a *stdout*

4::: `$ cat fit1 fit2 >& fit3` (fit1 existeix, fit2 no existeix)

totes les sortides estan a fit3 perquè redirigim *stdout* i *stderr* a fit3

1.6. filtres

Filtre: És un programa que rep dades per *stdin* i treu dades per *stdout*, sense modificar les dades entrades per *stdin*.

FILTRES

cat [-n] [*nom_fitxer*]

mostra stdin/*nom_fitxer*
-n numera les línies

head [-*num*] [*nom_fitxer*]

Mostra les primeres num línies del fitxer/stdin (10 per defecte)

tail [-*num*] [*nom_fitxer*]

Mostra les últimes *num* línies del fitxer/stdin (10 per defecte)
-f la sortida és dinàmica a mida que el fitxer canvia

wc [-lwc] [nom_fitxer]

compta línies, paraules i caràcters de *nom_fitxer*/stdin.
-l només el nombre de línies
-w només el nombre de paraules
-c només el nombre de caràcters

cut -c*llista* [*nom_fitxer*]

extreu les columnes citades a *llista*
format de *llista*:
A,*B* seleccionar columnes/camps *A* i *B*
A-*B* seleccionar columnes/camps des d'*A* fins a *B*
A- des de la columna/camp *A* fins al final
-B *des del principi fins a la columna/camp B*

cut -f*llista* -d*sep* [*nom_fitxer*]

extreu els camps citats a *llista* segons el separador *sep*
El format de la l*lista* és igual a l'anterior

grep [-cinv] *patró* [*nom_fitxer*]

Recerca de les línies dels fitxers/stdin del patró determinat
-c només mostra el número de línia
-i ignora majúscules/minúscules
-n afegeix el número de línia
-v mostra les línies que no contenen el *patró*

Format de *patró* (expressions regulars bàsiques):
. qualsevol caràcter simple
[] conjunt de caràcters
[^] qualsevol caràcter no inclòs als corxets
[-] rangs
* 0 o més ocurrències de l'expressió precedent
+ 1 o més ocurrències de l'expressió precedent
^*exp* qualsevol cadena que comenci amb *exp*
exp$ qualsevol cadena que acabi amb *exp*

tr *c1 c2* [*nom_fitxer*]

tradueix *c1* per *c2* de nom_fitxer/stdin

tr -s *c1* [*nom_fitxer*]

converteix c1 consecutius en un de sol

sed 's/*expr1*/*expr2*/[g]'

Substitueix expr1 per expr2
g substitueix totes les ocurrències

sed -r 's/*expr1*/*expr2*/[g]'

FILTRES

Substitueix expr1 per expr2 amb expressions regulars complexes
g substitueix totes les ocurrències

sort [-ndutsep] [-k num] [nom_fitxer]

Ordenar les línies de *nom_fitxer*/stdin
-n ordenació numèrica
-d no té en compte caràcters que no siguin
    lletres, números o blancs
-u no té en compte les línies duplicades
-t*sep* especifica un delimitador de camp
-k*num* especifica que ordenarem pel camp *num*

uniq [nom_fitxer]

converteix diverses línies iguals consecutives de *nom_fitxer*/stdin en una de sola

tee [-a] nom_fitxer1 [nom_fitxer2]

Mostra per pantalla nom_fitxer2/stdin i ho escriu a nom_fitxer1
-a enlloc de sobreescriure nom_fitxer1 afegeix
    a continuació

::exemples::

1::: Del fitxer /var/log/messages mostra:
  a:: el contingut de tot el fitxer
      $ cat /var/log/messages
  b:: les 3 primeres línies
      $ head -3 /var/log/messages
  c:: les últimes línies a mida que va canviant
      $ tail -f /var/log/messages
  d:: el número de línies
      $ wc -l /var/log/messages

2::: Del fitxer /etc/passwd mostra:
  a:: les tres primeres lletres dels usuaris del sistema
      $ cut -c1-3 /etc/passwd
  b:: el nom i el directori de treball dels usuaris
      $ cut -d: -f1,6 /etc/passwd

3::: Mostrar els usuaris que compleixin el següent:
  a:: el seu nom comenci per m o M
      $ grep -i ^m /etc/passwd
  b:: el seu id estigui entre el 1000 i el 1999
      $ grep ^.*:x:1...: /etc/passwd
  c:: tinguin per shell /bin/bash
      $ grep :/bin/bash$ /etc/passwd
  d:: tinguin qualsevol altre shell
      $ grep -v :/bin/bash$ /etc/passwd

4::: Mostra per pantalla el contingut del fitxer /etc/passwd:
  a:: amb els camps separats per un -
      $ tr ":" "-" /etc/passwd
  b:: reduint totes les comes a una
      $ tr -s "," /etc/passwd
  c:: canviant la shell /bin/bash per /bin/sh
      sed 's/\/bin\/bash/\/bin\/sh/' /etc/passwd
  d:: ordenat pel nom d'usuari
      sort /etc/passwd

e:: ordenat per l'identificador d'usuari
sort -nt: -k3 /etc/passwd

1.7. canonades (pipelines)

Canonada: Ens permetran redireccionar la sortida d'una comanda com a entrada d'una altra comanda.

COMANDES	
comanda1* | *comanda2	redireccionem *stdout* de *comanda1* a *stdin* de *comanda2*

::exemples::

1::: Mostrar els noms del*s usuari*s donats d'alta en el sistema i la seva shell d'inici, ordenats alfabèticament i separant els dos camps per un tabulador.
$ cat /etc/passwd | cut -f1,7 -d: | sort | tr ":" "\t"

2::: Visualitzar un número que sigui la suma de les línies dels fitxers /etc/profile i /etc/hosts
$ wc -l /etc/profile /etc/hosts | tail -1 | cut -f2 -d" "

3::: Comptar el número de fitxers (excloent directoris) del directori actual modificats el gener, alhora que s'emmagatzema el nom dels fitxers obtinguts en un altre fitxer anomenat "gener".
$ ls -l | grep -v ^d | tr -s " " | cut -f6,8 -d" " |
grep ^....-01- | cut -f2 -d" " |tee gener | wc -l

4::: A partir del fitxer /etc/passwd crear un fitxer anomenat grupmeu que estigui format pel nom d'usuari, shell inicial i directori inicial del*s usuari*s del teu mateix grup.
$ cat /etc/passwd | grep ^.*:.*:.*:`id -g`: | cut -f1,7,6 -d: > grupmeu

5::: Obtenir un fitxer anomenat procsroot que contingui l'identificador (PID) i el nom (CMD) de tots els processos que pertanyen a l'usuari* root.
$ ps -ef | grep "^root " | tr -s " " | cut -f2,8 -d" " > procsroot

6::: Obtenir un fitxer anomenat propsetc que contingui una llista (sense repeticions) amb els noms del*s propietari*s i grup al*s quals pertanyen els fitxers i directoris del directori /etc.
$ ls -l /etc | tr -s " " | cut -f3,4 -d" " | sort | uniq > propsetc

2. programació de shell scripts

2.1. execució de shell scripts

Shell script: No és més que un programa que usa el llenguatge propi de la shell. Hi ha diferents maneres d'executar-lo:

1. En una subshell de la shell activa
   a) $ bash *nom_script*
   b) $ *./nom_script* (necessitem permisos d'execució)
   c) $ *nom_script* (necessitem permisos d'execució i que el directori on està *nom_script* estigui al PATH)

2. Com un programa executable que s'executa com a procés fill de la shell
   Com a primera línia de l'script posarem amb quina shell volem que ens l'interpreti:
   ```
   #!/bin/bash
   ```
   i executem l'script com en el cas 1

3. En la mateixa shell activa
   $ source *nom_script*
   $. *nom_script*

::exemple::

Crea un script de nom dorm.sh amb la següent línia:

```
sleep 222
```

Execució tipus 1:

```
$ chmod u+x dorm.sh
$ ./dorm.sh

en una altra consola veuríem:
$ pstree -p
   ...
   -bash(905)---bash(1542)---sleep(1543)
   ...
```

Execució tipus 2:

```
Ara l'script tindrà el següent contingut:
#!/bin/bash
sleep 222

$ ./dorm.sh
en una altra consola veuríem:
$ pstree -p
   ...
   -bash(905)---dorm.sh(1564)---sleep(1565)
   ...
```

Execució tipus 3:

```
$ . dorm.sh
en una altra consola veuríem:
$ pstree -p
   ...
   -bash(905)---sleep(1576)
   ...
```

2.2. depuració de shell scripts

Depuració: Quan fem un programa i els resultats que aquest produeix no són els esperats és molt útil tenir una eina que ens permeti depurar-lo. És a dir, una eina que ens permeti fer un seguiment pas a pas de per on va el flux del programa.

DEPURACIÓ	
set -x	activa la depuració
set +x	desactiva la depuració
set -e	surt immediatament si una ordre simple acaba sense èxit (amb un codi de retorn diferent de zero)
set +e	desactiva el sortir si una ordre simple acaba sense èxit

::exemple::

Si tenim el següent script:

```
SALUDA=”hola”
DESPEDEIX=”adeu”
set -x
echo “$SALUDA $LOGNAME”
echo “Usuaris Connectats”
who
set +x
echo “$DESPEDEIX $LOGNAME”
```

quan l'executem ens depurarà les línies de codi que estan entre “set -x” i “set +x” i tindrem el següent resultat per pantalla:

```
++ echo hola hm
hola hm
++ echo Usuaris Connectats
Usuaris Connectats
++ who
hm   tty1         Jun  3 19:39
++ set +x
adeu hm
```

2.3. comentaris

Comentaris: Quan programem hem de tenir present que no estem sols al món o, que encara que ho estiguem, la memòria ens pot fallar i no entendre el programa que nosaltres mateixos hem

escrit... En resum, que hem de comentar els nostres scripts per fer-nos a tots la vida més fàcil :)

COMENTARIS	
# comentari	comentar parts de codi

2.4. paràmetres i variables especials

Paràmetres: Un paràmetre d'un script seran tots aquells valors que adjuntem quan executem aquest i que volem poder veure dins del programa.

$ *./nom_script paràmetre1 paràmetre2 ... paràmetren*

Podrem fer referència als paràmetres passats segons la següent taula:

PARÀMETRES	
$0	nom del shell-script que s'està executant
$*n*	paràmetre passat al shell-script en la posició *n*
"$*"	expandeix els paràmetres en una cadena: "par1 par2 ..."
"$@"	expandeix els paràmetres en cadenes diferenciades: "par1" "par2" ...
shift *n*	Desplaçar els paràmetres en *n* posicions. Sense paràmetre n desplaça 1 posició.

Variables especials: Tenim diferents variables definides en qualsevol script que podem usar segons les nostres necessitats.

VARIABLES ESPECIALS	
$#	número de paràmetres
$$	PID de la shell procés que s'està executant
$!	PID de l'últim procés executat
$?	Codi de retorn de l'últim procés executat

::exemple::

(sistema.sh) script que ens informa de diverses dades del sistema

2.5. instruccions e/s

Instruccions d'entrada i sortida: Són aquelles instruccions que ens permetran llegir dades de *stdin* (entrada) i mostrar dades per *stdout* (sortida)

COMANDES	
read *nom_var*	entrada per *stdin*
echo [-ne] *cadena/$nom_var*	sortida per *stdout* amb salt de línia final -e interpreta caràcters amb \ \n : salt de línia \t : tabulador ... -n suprimeix el salt de línia final

::exemple::

```
$ read varmeva              llegim una variable de teclat
3
$ echo -e "La meva var és:\t\t$varmeva"        miro el seu
La meva var és:          3                     valor amb un
                                               format
```

2.6. operadors

Operadors numèrics: són aquells que ens permeten operar amb números i variables que continguin números.

Operadors lògics: són aquells que ens permeten especificar condicions compostes

OPERADORS NUMÈRICS		OPERADORS LÒGICS	
+	suma	!	no
-	resta	&&	i
*	producte	||	o
\/	divisió		
%	mòdul		
\(\)	parèntesis		

2.7. avaluació d'expressions numèriques

En moltes ocasions haurem de fer càlculs numèrics ja sigui per mostrar el resultat o bé per emmagatzemar-lo en una variable.

EXPRESSIONS NUMÈRIQUES	
expr *expr_num*	avalua *expr_num* traient el resultat per *stdout* (a *expr_num* hem de separar operadors d'operands amb un espai)
let *expr_num* **((*expr_num*))**	avalua *expr_num* (a *expr_num* no hem de separar operadors d'operands amb un espai, ens serveix per assignar)

::exemples::

```
1::: $ expr 3 + 5          calculem 3+5
     8                     8!

2::: $ ((a=3+5))           calculem 3+5 i posem el resultat a la variable a
     $ echo $a             mostrem el valor de la variable
     8
```

```
3::: $ a=1
     $ ((a=$a+1))        incrementem el valor d'a
     $ echo $a           mostrem el valor de la variable
     2
```

2.8. especificació de condicions

Condició: Per tal de trencar el flux d'un programa necessitem especificar condicions que ens el bifurquin cap a un cantó o cap a un altre. Per especificar condicions s'usa:

test *expr*

o bé

[expr]

A continuació s'exposen les condicions que podem especificar:

CONDICIONS FITXERS	
[-e *fit*]	true si el fitxer existeix
[-d *fit*]	true si el fitxer existeix i és un directori
[-f *fit*]	true si el fitxer existeix i és regular
[-L *fit*]	true si el fitxer existeix i és un enllaç simbòlic
[-r *fit*]	true si el fitxer existeix i té permís de lectura
[-w *fit*]	true si el fitxer existeix i té permís d'escriptura
[-x *fit*]	true si el fitxer existeix i té permís d'execució
[*fit1* -nt *fit2*]	true si fitxer1 és més nou que fitxer2
[*fit1* -ot *fit2*]	true si fitxer1 és més antic que fitxer2
CONDICIONS CADENES	
[*cad*]	true si no és la cadena buida
[-n *cad*]	true si la longitud de *cadena* és diferent a 0
[-z *cad*]	true si la longitud de *cadena* és 0
[*cad1* = *cad2*]	true si *cadena1* i *cadena2* són iguals
[*cad1* != *cad2*]	true si *cadena1* i *cadena2* són diferents
CONDICIONS ENTERS	
[*num1* -eq *num2*]	true si *num1* i *num2* són iguals
[*num1* -ne *num2*]	true si *num1* i *num2* són diferents
[*num1* -gt *num2*]	true si *num1* és més gran que *num2*
[*num1* -ge *num2*]	true si *num1* és més gran o igual que *num2*
[*num1* -lt *num2*]	true si *num1* és més petit que *num2*
[*num1* -le *num2*]	true si *num1* és més petit o igual que *num2*

2.9. estructures alternatives

Estructures alternatives: Són aquelles que ens permeten executar un tros de codi segons si es compleix o no una condició.

ESTRUCTURA if	ESTRUCTURA case

if *condicio1*
then
  instruccions
elif *condicio2*
then
  instruccions
else
  instruccions
fi
	case $*nom_var* in
	  patro1) *instruccions*;;
	  patro2) *instruccions*;;
	  ...
	  patron) *instruccions*;;
	  *) *instruccions*;;
	esac

::exemples::

1::: (existrc.sh) script que comprova l'existència del fitxer .bashrc

2::: (propsfit.sh) script que comprova si existeix el fitxer passat per paràmetre i, si existeix, ens dóna les seves propietats

3::: (backup.sh) script que crea una còpia de seguretat del directori que li passem com a paràmetre i la guarda a /var/backups amb nom de format backup-aaaammdd.tgz

4::: (soinifi.sh) script pensat per a posar a /etc/init.d que ens reproduirà un so en entrar al sistema i un so en sortir-ne.

2.10. estructures iteratives

Estructures iteratives: Són aquelles que ens permeten executar diversos cops un tros de codi.

ESTRUCTURA while	ESTRUCTURA until	ESTRUCTURA for

while *condicio*
do
  instruccions
done
	until *condicio*
	do
	  instruccions
	done
		for *nom_var* in *llista_valors*
		do
		  instruccions
		done

::exemples::

1::: (rename.sh) script que reanomena diversos fitxers alhora en el directori actual.

2::: (llistafit.sh) script que accepta com a arguments noms de fitxers i mostra el contingut de cadascun d'ells precedit d'una capçalera amb el nom del fitxer.

3::: (estatdperm.sh) script que realitza un estudi de tot l'arbre de directoris i fitxers a partir del directori

passat com a paràmetre de forma que obtinguem la següent info:

- Número de fitxers dels quals disposem de permís de lectura
- Número de fitxers dels quals disposem de permís d'escriptura
- Número de fitxers dels quals disposem de permís d'execució
- Número de fitxers dels quals no tenim permís de lectura
- Número de fitxers dels quals no tenim permís d'escriptura
- Número de fitxers dels quals no tenim permís d'execució

4::: (grafica.sh) script que rep com a arguments números compresos entre 1 i 75. Donarà error en cas que algun argument no estigui dins del rang i acabarà sense fer res. En cas contrari generarà una línia per a cada argument amb tants asteriscos com indiqui el seu argument.

5::: (onesta.sh) script que busca la presència de la comanda passada com argument en algun dels directoris referenciats a la variable PATH, senyalant la seva localització i una breu descripció de la comanda.

6::: (fitpar.sh) script que posa cadascuna de les paraules passades per paràmetre en un fitxer. Aquests fitxers s'anomenaran paraula0, paraula1,... respectivament.

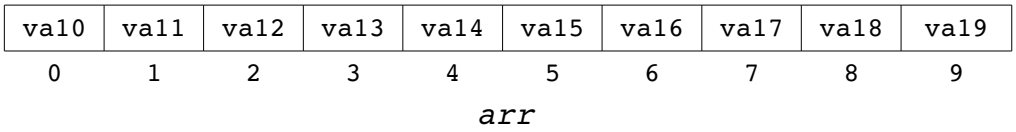
7::: (sequencia.sh) script que compta per tu :-)

8::: (binbash.sh) script que afegeix la línia #!/bin/bash al principi dels fitxers que els hi passis com a paràmetre, si és que ja no el té (per a gent amb poca memòria...)

3. una mica més...

3.1. taules

Taula: És una estructura de dades composta.



A continuació podem veure com definir-la i com fer-hi referència:

TAULES	
nom_arr=(val1 val2 ... valn)	declaració i assignació inicials
declare -a *nom_arr*	declaració d'una taula. Útil per fer assignacions dinàmiques.
nom_arr[index]=val	assignació de *val* a l'element *index* de *nom_arr*
${*nom_arr*[*index*]}	fer referència a l'element situat a *index* de la taula *nom_arr*

Podem fer referència a elements i característiques de la taula amb les següents expansions:

EXPANSIONS	
${*nom_arr*[#]}	número d'elements de la taula
${*nom_arr*[@]}	llista els elements de la taula tractats cadascun d'ells com a una cadena
${*nom_arr*[*]}	llista els elements de la taula tractats com a una única cadena
${#*nom_arr*[*index*]}	longitud de ${*nom_arr*[*index*]}

::exemple::

(gen_menu.sh) script que admet com a arguments parelles formades per 'descripció' i 'comanda' i que construeix un menú d'opcions on qualsevol de les comandes pot ser executada seleccionant la descripció corresponent.

3.2. funcions

Definició d'una funció: Per definir una funció tenim dues possibilitats:

```
nom_funcio(){                    function nom_funcio{
   ...                              ...
   instruccions                     instruccions
   ...                              ...
}                                }
```

Paràmetres: Per fer referència als paràmetres que ens puguin arribar a la funció ho farem com hem explicat a l'apartat 2.4. ($1, ..., $n,...). Tots els paràmetres es passen per valor, és a dir quan retornem de la funció el valor dels paràmetres no haurà canviat.

Retorn: Podem fer que les funcions retornin un valor.

RETORN	
return *valor*	interromp la funció assignant un valor al codi de retorn de la funció

::exemples::

1::: (menu.sh) menú amb diferents opcions

2::: (gen_aniv.sh) Suposem que tenim un fitxer anomenat "aniversari.txt" les línies del qual tinguin el següent format:

nom:dataaniversari

Realitzem un script anomenat "generaaniv" que et generi el fitxer que hauríem de passar a la comanda crontab per tal que el dia abans de cada data d'aniversari rebis un mail que tingui la següent informació:

```
...
Subject: Recordatori Aniversari
...
Demà dia [data] és l'aniversari de [nom]. No oblidis felicitar-l@.
```

3.3. expressions regulars

Expressions regulars: Són una eina per cercar coincidències entre un text i un patró. L'explicació donada aquí està basada en les expressions regulars estil Perl.

COMODÍ	
.	qualsevol caràcter
CLASSES DE CARÀCTERS	
[*llista*]	algun dels caràcters de la llista
[*min-max*]	caràcters compresos entre *min* i *max*
[^*llista*]	qualsevol caràcter que no estigui a la llista
\w	paraula
\W	contrari de \w
\s	espais, tabuladors,... (espai, \t, \n, \r)
\S	contrari de \s
\d	dígit
\D	contrari de \d
\A	començar a mirar pel principi de la cadena
\Z	començar a mirar pel final de la cadena

\b	concordar amb els límits de la paraula
\B	contrari de \b
[[:alnum:]]	alfanumèrics
[[:alpha:]]	alfabètics
[[:cntrl:]]	caràcters de control
[[:digit:]]	dígits
CLASSES DE CARÀCTERS	
[[:graph:]]	gràfics
[[:lower:]]	minúscules
[[:print:]]	caràcters imprimibles
[[:punct:]]	caràcters de puntuació
[[:space:]]	espais
[[:upper:]]	majúscules
[[:xdigit:]]	dígits hexadecimals
ALTERNATIVES	
alter1*|*alter2	pot aparèixer *alter1* o *alter2*
QUANTIFICADORS	
?	0 o 1 ocurrència de l'expressió precedent
*	0 o més ocurrències de l'expressió precedent
+	1 o més ocurrències de l'expressió precedent
{*m*}	*m* ocurrències de l'expressió precedent
{*m*,}	*m* ocurrències o més de l'expressió precedent
{*m*,*n*}	de *m* a *n* ocurrències de l'expressió precedent
??, *?, +?, {}?	el mateix però no intenta agafar el màxim de caràcters (no 'greedy')
ÀNCORES	
^*expr*	qualsevol cadena que comenci amb *exp*
***expr*$**	qualsevol cadena que acabi amb *exp*
GRUPS I REFERÈNCIES	
(*expr*)	fer un grup per poder-lo referenciar després
n	referència al grup n-èsim
(?:*expr*)	grup sense referència

:::exemples:::

1::: [a-z] lletres minúscules
[A-Z] lletres majúscules
[0-9] números
[,'¿!¡;:\.\?] caràcters de puntutació
[A-Za-z] lletres de l'alfabet
[a-Z] el mateix que l'anterior
[A-Za-z0-9] caràcters alfanumèrics
[^a-z] tot menys les lletres minúscules
[^0-9] tot menys números

2::: a.b axb aab abb aSb a#b ...

```
a..b          axxb aaab abbb a4$b ...
[abc]         a b c
[aA]          a A
[aA][bB]      ab Ab aB AB
[0-9][0-9][0-9]     000 001 009 010 019 100 999
[0-9]*        cadena_buida 0 1 9 00 99 123 456 999 9999
[0-9][0-9]*         0 1 9 00 99 123 456 999 9999 99999 99999999
^.*$          qualsevol línia complerta
[0-9]+        0 1 9 00 99 123 456 999 9999 99999 99999999 ...
[0-9]?        cadena_buida 0 1 2 3 ...
^a|b          a b
(ab)*         cadena_buida ab abab ababab ...
^[0-9]?b      b 0b 1b 2b 9b
([0-9]+ab)*         cadena_buida 1234ab 9ab9ab9ab 9876543210ab
                    99ab99ab

3::: ([0-9]) \1 ([0-9]) 3 3 5
     el cargol|gat      el cargol el gat
     ho{1,4}la          hola hoola hooola hoooola
     (slashdot|barrapunto)      slashdot barrapunto
     s[aeou]c           sac, sec, soc, suc
     -?[0-9]+           64, -2, 65536, -512
     0x[0-9A-F]{2}              0xF0, 0x3B, 0xAA
     go{2,5}ggle                google, gooogle, goooogle, gooooogle
     \bkernel\b         kernel, pero no kernelpanic o microkernel
     \d+\.\d+\.\d+\.\d+ 192.168.0.1 (direcció ip)
     (?:\d+\.){3}\d     192.168.0.1 (direcció ip)
     (?:[\w-]+\.)+(?:net|com|org)       ii.jocs.fractals.net
     s/<i>(.*?)<\/i>/<b>\1<\/b>/g       itàliques  =>  negretes  (en
     document html)
     s/(\d+)\.(\d+)\.(\d+)\.(\d)+/\4.\3.\2.\1/  192.168.0.1 =>
                                                1.0.168.192
     (\d+\.\d+\.\d+\.\d+)(?:\n)*\1(?:\n)*\1     3 cops la mateixa ip
     (\d+\.\d+\.\d+\.\d+\n*){3}         el mateix que l'anterior
```

3.4. comandes mooooolt útils

3.4.1. sed

sed: és un editor no interactiu, rep l'entrada per *stdin* o per un fitxer, fa les operacions que pertoquin i treu el resultat per *stdout*. De totes les seves funcionalitats detallarem aquí tres de les més potents: escriure, esborrar i substituir. La sintaxi és:

sed [*opcions*] {*operador* | *script fitxer*}

OPCIONS	
-r	per poder usar expressions regulars complexes. Per exemple, per poder fer referència a grups: podem utilitzar *n* per fer referència a l'*n*-èsim grup, on cada grup es delimita per (expr).
-f	si volem usar un script enlloc d'un operador
-n	no escriu per pantalla la línia que està tractant

OPERADORS BÀSICS	
***rang*p**	imprimeix les línies de *rang*
***rang*d**	esborra les línies de *rang*
s/*patro1*/*patro2*/[*modificador*]	substitueix la primera ocurrència de *patro1* per *patro2*
***rang*/s/*patro1*/*patro2*/[*modificador*]**	substitueix la primera ocurrència de *patro1* per *patro2* de les línies de *rang*

MODIFICADORS	
g	substitueix a totes les ocurrències de les línies, no només a la primera
i	no distingeix entre minúscules i majúscules

:::exemples:::

8d	Esborra la 8a línia
/^$/d	Esborra totes les línies en blanc
/kernel/p	Imprimeix les línies amb la paraula kernel (amb -n)
s/Windows/Linux/	Substitueix "Linux" per la primera instància de “Windows" a cada línia
s/Windows/Linux/g	Substitueix "Linux" per cada instància de “Windows”
s/ *$//	Esborra tots els espais del final de cada línia
s/00*/0/g	Redueix totes les seqüències de 0's a un únic 0
/Windows/d	Esborra totes les línies on apareix Windows
s/Windows//g	Esborra totes les paraules Windows, sense esborrar la línia

:::exemples:::

```
sed -r 's/(La|El) (.*) és.*/\2/'
```

Extreu el subjecte (sense article) de frases del tipus *“La pilota petita és d'un color molt estrany”*

```
sed -r 's/(La|El) ([^ ]*) (.*) (és.*)/\1 \3 \2 \4/'
```

Posa la primera paraula del subjecte al final d'aquest

3.4.2. awk

És un llenguatge de programació dissenyat per processar dades basades en text, ja siguin fitxers o fluxos de dades.

La invocació de awk es composa d'un programa i d'un o més fitxers. Cadascuna de les accions del programa s'executa per a cada línia del/s fitxer/s.

awk [-F *separador*] [-v *var=valor*] [-f *prog_awk*|'*prog_awk*'] [*fitxers*]

OPCIONS	
-F	separador de camps per defecte
-v var=*valor*	definir valors inicials a variables abans de l'execució del programa
-f *prog_awk*	fitxer que conté el programa a executar

Un programa awk estar composat per una o diverses estructures amb la següent sintaxi:

expressió { *acció1*; *acció2*; ... }

Expressió: expressió lògica o regular

Accions: Si l'avaluació de l'expressió és positiva s'executa la/les acció/ns indicada/es

ACCIONS	
print	Imprimeix per pantalla

:::exemple:::

```
$ echo -e "\nun\ndos\ntres" | awk '{ print "Jo uso"; print
"Linux" }'
Jo uso
Linux
Jo uso
Linux
Jo uso
Linux
```

Per a cada línia, com que l'expressió es compleix, ens fa les accions: imprimir "Jo uso" i imprimir "Linux"

Plantilles: Enlloc de l'expressió podem posar unes plantilles que ens serveixen perquè s'executi alguna/es acció/ns abans i/o al finalitzar el programa:

PLANTILLES DE PRÒLEG I EPÍLEG	
BEGIN	L'acció s'executarà al principi
END	L'acció s'executarà al final

Per tant, la sintaxi d'un programa awk serà:

```
BEGIN {accions_inicials}
expressio1 {accions}
expressio2 {accions}
...
END {accions_finals}
```

Camps: Cada línia del fitxer que li passem a awk està formada per camps (si no li passem un delimitador, aquest serà l'espai en blanc).

CAMPS	
$*n*	n-èsim camp

:::exemples:::

1::: Imprimir camps:

```
$ echo “un dos tres” | awk '{ print $1,$2 }'
un dos
```

Imprimim el 1er i 2on camp, separats per un espai

2::: Extreure d'un fitxer les línies on apareix la paraula Linux:

```
$ awk '/linux/' fitxer.txt
```

3::: Posar l'extensió .nou als fitxers que tenen l'extensió .new:

```
$ ls -l *.new | awk '{print "mv "$8" "$8".nou"}' | bash
```

4::: Esborrar tots els directoris, però no els fitxers:

```
$ ls -l | grep '^d' | awk '{print "rm -r "$8}' | bash
```

5::: Esborrar tots els fitxers, però no els directoris:

```
$ ls -l | grep -v ^d | awk '{print "rm "$8}' | bash
```

6::: Suma de números positius:

sumaawk.txt:

```
BEGIN { total=0 }          inicialitzem la variable total a 0
$1>0{ total=total+$1 }     si el número és positiu sumem
                           el primer camp
END {print "El total és",total}  imprimim el resultat
                                 final
```

```
$ echo -e "1\n2\n-4\n4" | sumaawk -f awk.txt
El total és 7
```

Variables: Com a qualsevol altre llenguatge de programació podem definir variables. No s'han de declarar, en tenim prou assignant-hi un valor.

Tenim a més, un conjunt de variables predefinides pel propi awk:

VARIABLES PREDEFINIDES	
FS	conté el caràcter delimitador de camps (per defecte, espai en blanc)
RS	conté el caràcter que indica on s'acaba cada registre (per defecte, \n)
OFS	conté el separador de camps per a la sortida generada (per defecte, espai en blanc)
ORS	conté el caràcter de final de registre per a la sortida generada (per defecte, \n)
NF	conté el número total de camps del registre que s'està processant
NR	conté el número d'ordre del registre que s'està processant

:::exemples:::

1:::

```
$ echo –e "Uso el sistema\noperatiu anomenat GNU/Linux" |
awk –v OFS="\t" –v ORS=“\t” ‘{print $1, $2, $3}'
Uso    el    sistema    operatiu    anomenat    GNU/Linux
```

Com a sortida, la separació de camps i la de registres és un tabulador

2::: awk.txt:

```
{ print "Processant la línia", NR }
```

imprimim el número de línia que estem processant

```
NF > 1 { print "La línia té més d'una paraula" }   mirem si té més d'un camp
$1 == "GNU/Linux" {print "La primera paraula és GNU/Linux"}
        mirem si el primer camp és GNU/Linux

$ echo -e "GNU/Linux es un SO\nM'agrada
 GNU/Linux\nGNU/Linux és el meu SO" | awk -f awk.txt
Processant la línia 1
La línia té més d'una paraula
La primera paraula és GNU/Linux
Processant la línia 2
La línia té més d'una paraula
```

Operadors: Per poder operar tenim els següents operadors:

OPERADORS			
numèrics		**comparació**	
+	suma	**>**	major que
-	resta	**>=**	major o igual que
*	multiplicació	**<**	menor que
/	divisió	**>=**	menor o igual que
%	residu	**==**	igual
^		**!=**	diferent
var++	incrementar var en 1		
var--	decrementar var en 1		
lògics		**concatenació**	
&&	i	**espai**	concatenació
||	o		
!	negació		

Estructures de control: Disposem també de les diferents estructures de control de qualsevol llenguatge de programació. En el cas de awk s'han heretat la sintaxi del llenguatge de programació C.

EST. if	**EST. for**	**EST. while**	**EST. do/while**

if(*cond*) {
  instruccions
}
[else {
  instruccions
}]
	for (inic;cond;instr){
	  instruccions
	}
		while(cond){
		  instruccions
		}
			do{
			  instruccions
			} *while(cond)*

:::exemples:::

1::: posnegawk.txt:

```
{
   if( $1 > 0 ){            mirem si el primer camp és positiu...
      print $1,"és positiu";
   }
   else{                    ...o negatiu
```

```
      print $1,"és negatiu";
   }
}

$ echo -e "12\n-15\n10" | awk -f posnegawk.txt
12 és positiu
-15 és negatiu
10 és positiu
```

2::: taulaawk.txt:

```
{
   print "\nTaula del",$1
   print "-------------"
   for( i = 1; i < 11; i ++ ){
      j = $1 * i;
      print i,"*",$1,"=",j;
   }
}

$ echo -e "2\n3" | awk -f awk.txt

Taula del 2
-------------
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14
8 * 2 = 16
9 * 2 = 18
10 * 2 = 20

Taula del 3
-------------
1 * 3 = 3
2 * 3 = 6
3 * 3 = 9
4 * 3 = 12
5 * 3 = 15
6 * 3 = 18
7 * 3 = 21
8 * 3 = 24
9 * 3 = 27
10 * 3 = 30
```

Funcions: Tenim a més diverses funcions predefinides:

FUNCIONS	
length(*s*)	Retorna la longitud de *s* en bytes
rand()	Retorna un número a l'atzar entre 0 i 1
srand()	Inicia la llavor de generació a l'atzar
int(*var*)	Retorna *var* convertit en un enter
substr(*s*,*m*,*n*)	Retorna la subcadena de *s* començant per la posició *m* amb una longitud de *n*
index(*s*,*t*)	Posició de *s* on apareix *t* o 0 si no està
match(*s*,*r*)	Posició de *s* on es compleix l'expressió *r*
split(*s*,*a*,*fs*)	retorna *s* en elements separats per *fs* a la taula *a*
sub(*r*,*t*,*s*)	Canvia en *s* la cadena *t* per *r*
gsub(*r*,*t*,*s*)	Canvia en *s* la cadena *t* per *r* en totes les ocurrències
sprintf(*f*,*e*,*e*,...)	Imprimeix amb format
system(*cmd*)	Executa *cmd* i retorna el codi de retorn
tolower(*s*)	Retorna *s* en minúscules
toupper(*s*)	Retorna *s* en majúscules
getline	Força una lectura de fitxer

:::exemples:::

1::: Programa que imprimeixi les línies amb més de 72 caràcters:

```
length>72
```

2::: Substituir la paraula hola per adéu

```
{sub("hola","adeu");print}
```

3::: Imprimir línies al revés

```
{for (i=NF; i>0; i--)
   print $i}
```

4::: Imprimir les línies d'un fitxer suprimint les repetides consecutives

```
$1 != anterior {print; anterior=$1}
```

5::: Imprimir el codi ASCII

```
{ for (i=30; i<255; i++)
   printf("caracter %c\t%d\t%o\n", i, i, i) }
```

6::: Afegir els usuaris amb les seves contrasenyes (existents en un fitxer separats per un espai en blanc):

```
{ system ("useradd -p `mkpasswd ”$2”` "$1) }
```

3.5. una mica de color i moviment...

L'*American National Standards Institute* (*ANSI*) proporciona una sèrie de seqüències de caràcters per poder realitzar determinades tasques sota el S.O.. Aquí veurem les seqüències que ens permeten formatar els caràcters de la pantalla (negreta, color,...), moure el cursor, entre d'altres. Per a aconseguir-ho haurem de posar:

SEQÜÈNCIA_ESCAPAMENT+SEQÜÈNCIA_CONTROL

SEQÜÈNCIES D'ESCAPAMENT	
^[	Si fem un text directe i volem veure'l amb cat Per obtenir aquest caràcter amb l'editor VI: ctrl+x ESC
\E	Si fem un echo -e “”

SEQÜÈNCIES DE CONTROL	
ATRIBUTS DEL TEXT	
[0m	Text normal (reset)
[1m	Negreta
[3m	Cursiva
[4m	Subratllat
[5m	Intermitent
[7m	Invers
[22m	No negreta
[23m	No cursiva
[24m	No subratllat
[25m	No intermitent
[27m	No invers
COLORS	
[XX;YYm	XX: color de lletra YY: color de fons **COLORS DE LLETRA** / **COLORS DE FONS** **30** negre / **40** negre **31** vermell / **41** vermell **32** verd / **42** verd **33** groc / **43** groc **34** blau / **44** blau **35** magenta / **45** magenta **36** cian / **46** cian **37** blanc / **47** blanc Si combinem aquests valors amb 0 (text normal) i 1 (negreta) obtenim colors forts o suaus.
MOVIMENT DEL CURSOR	
[*x*A	Pujar *x* línies
[*x*B	Baixar *x* línies
[*y*C	Anar a la dreta y espais
[*y*D	Anar a l'esquerra y espais
[*y*;*x*H **[*y*;*x*f**	Posicionar el cursor a *y*,*x*
[?6h	Posicionar a la cantonada superior esquerra
[s	Gravar cursor i atributs
[u	Restaurar atributs gravats
CONTROL DE PANTALLA	
[2J	Netejar pantalla
[K	Esborrar fins fi de línia
[?5h	Encén invers
[?5l	Apaga invers

::exemples::

1::: Text vermell clar sobre verd subratllat i tornar a deixar els colors normals

```
echo -e “\E[1;4;31;42mHOLA\E[0m”
```

2::: Posicionar el cursor a la fila 15 columna 40 amb fons groc, text blau (colors suaus) i intermitent. Posar novament el cursor 5 línies més avall

```
echo -e "\E[2J\E[15;40H\E[1;5;34;43mHOLA\E[0m\E[5B"
```

3::: (testsuma.sh) Script que proposa deu sumes i et dóna la nota que treus. S'usen diferents colors i posicionaments del cursor.

4. annexos

4.1. exercicis

:::VARIABLES:::

1. Usant variables:
   a) Assignar el valor “Dilluns” a la variable DIA1, el valor “Dimarts” a la variable DIA2 i així consecutivament fins a la variable DIA7.
   b) Mostrar el valor de totes les variables per verificar.
   c) Usant aquestes variables obtenir la sortida:
      Setmana: Dilluns Dimarts Dimecres Dijous Divendres Dissabte Diumenge
   d) Usant només les variables definides, carregar a la variable SETMANA la llista de dies separats per espais.
2. Usant variables d'entorn, mostrar:
   a) camí del directori de treball de l'usuari
   b) nom de login de l'usuari
   c) nom de la terminal en ús de l'usuari
   d) nom de d'intèrpret de comandes actual
   e) camí o camins de cerca d'executables
   f) totes les variables d'entorn
3. Usant variables:
   a) Declara la variable VIES amb el valor “/usr/doc:/var/lib/dpkg”. Mostra el seu contingut.
   b) Agregar a la variable VIES el directori /usr/doc/HOWTO al final i /usr/DOC/FAQ al principi (separats tots els directoris amb :)
4. Quina és la sortida de les següents comandes?
   ```
   echo $LOGNAME
   echo “$LOGNAME”
   echo '$LOGNAME'
   echo \”$LOGNAME\”
   echo “El meu login és $LOGNAME”
   echo 'El meu login és $LOGNAME'
   ```
5. Usant variables:
   a) Visualitza el contingut de la variable on es guarda el prompt. Guarda el seu valor en una variable local que s'anomeni SAVE.
   b) Canviar el prompt per a que es mostri així:
      UNIX llest$
   c) Fes que el prompt torni a tenir el seu valor inicial usant la variable SAVE.
6. Usant variables:
   a) Inicialitzar la variable VAR1 amb la cadena “shell bash 1”. Mostrar el seu contingut. Invocar la shell csh. Quin valor té ara la variable VAR1? Per què?
   b) Surt de csh. Quin valor té ara VAR1? Per què?
   c) Fes que la variable VAR1 sigui una variable d'entorn i repeteix els passos a) i b) Què ha passat?
   d) Si en lloc d'invocar csh invoquem bash. Passa el mateix?
7. Afegeix el directori /usr/prog/bin a la variable PATH. Com

ho faries per comprovar que funciona bé el nou PATH?
8. Crea la variable MID que contingui els valors de les variables HOME i LOGNAME separats per dos punts.

:::REDIRECCIONAMENT:::

9. Crea l'arxiu *línies* amb el text "Arxiu línies" com a contingut. Agregar a l'arxiu creat una línia de text, per exemple "Aquesta és la línia 1" sense usar cap editor de text.
   a) Amb la comanda cat, mostra per pantalla el contingut de l'arxiu /etc/services
   b) Escriu la comanda cat redireccionant l'entrada estàndard des de l'arxiu /etc/services i la sortida estàndard cap a l'arxiu serveis.txt. Visualitza serveis.txt
   c) Usant echo, crea l'arxiu errors.txt amb el contingut "Arxiu d Errors"
   d) Amb la comanda cat intenta mostrar l'arxiu noexist.xxx sense redireccionar entrada estàndard, però redireccionant la sortida estàndard cap a noexist.txt i l'error estàndard cap a agregar a l'arxiu errors.txt. Visualitza noexist.txt i errors.txt.
10. Utilitza la següent sentència " $ cat f1 f2 ". El fitxer f1 ha d'existir i el f2 no ha d'existir. Observa la sortida pel monitor.
   a) Aconsegueix que la sortida d'errors vagi a un fitxer anomenat errors.
   b) Aconsegueix que el contingut del fitxer que existeix es copii a un fitxer anomenat f3 (a més del que has aconseguit a l'apartat a)).
11. Usant redireccionament, crea un fitxer que contingui la següent informació:
   - Una línia amb el contingut "Informe del sistema"
   - Dos línies en blanc
   - El calendari del mes actual
   - La data actual
   - El tipus d'ordinadors que estem usant
   - Els usuaris que estan connectats

:::FILTRES I PIPELINES:::

12. Donat el següent fitxer anomenat text:

```
a
aa
ab
aba
aaa
abab
abba
La cadena a es capicua
Tambe es capicua la cadena aa
La cadena ab no es capicua
En una cadena capicua el seu final reflexa el seu principi
Si concatenes una cadena i el seu reflex el resultat es capicua
A
```

AA
ABA

a) Llistar les línies que continguin una lletra ‘a’
b) Llistar les línies que continguin dos lletres ‘a’ consecutives
c) Llistar les línies que no continguin la lletra ‘a’
d) Llistar les línies que no continguin lletres majúscules
e) Llistar les línies que comencin per la lletra ‘a’
f) Llistar les línies que comencin per una lletra minúscula
g) Llistar les línies que acabin amb la cadena ‘capicua’

13. Llistar tots els arxius del directori actual sense que apareguin els directoris.
14. Mostrar tots els processos del sistema amb UID root.
15. Mostrar tots els processos del sistema amb UID diferent de root.
16. Extreure els noms dels usuaris del resultat de l'ordre “who”.
17. Extreure els camps 1 i 3 del resultat de l'ordre “who”.
18. Extreure els permisos de tots els fitxers del directori $HOME.
19. Llistar el propietari i mida de tots els fitxers del directori $HOME.
20. El fitxer /etc/passwd del servidor Linux conté informació de tots els usuaris que tenen un compte a la màquina. Cada línia correspon a un usuari diferent i en ella apareixen els següents camps delimitats per “:”

Nom d'usuari
Password codificada en forma de x
Identificador de l'usuari
Identificador del grup al qual pertany l'usuari
Informació respecte l'usuari (nom, cognoms, etc.)
Directori de treball
Intèrpret de comandes utilitzat

Un exemple de la línia corresponent a un usuari en aquest fitxer seria:

maria:x:210:204:Maria Sanchez:/home/maria:/bin/bash

a) Comprova si existeix o no una entrada del teu usuari en aquest fitxer.
b) Llista tots els usuaris del mateix grup que tu que existeixin en aquest fitxer.
c) Llista els identificadors d'usuari.
d) Llista els shells usats pels usuaris que tinguin el mateix grup que tu.
e) Llista els camps 1 i els camps 3 a 5.

21. Mostrar el contingut del fitxer /etc/passwd ordenat alfabèticament.
22. Ordenar alfabèticament el resultat de l'ordre “who”
23. Extreure els noms dels usuaris del resultat de l'ordre “who” i ordenar-los alfabèticament.
24. Extreure els noms dels usuaris del resultat de l'ordre “who”, ordenar-los alfabèticament i eliminar els noms repetits.
25. Llistar les mides i els noms dels fitxers (excloent els directoris) del directori actual ordenats per mida.
26. En el fitxer /etc/group s'especifiquen els grups existents

al sistema amb el seu identificador i els seus components. El format de cada línia és el següent:

nom_grup:id_grup:comp1,comp2,...compn

a) Fes sortir per pantalla les línies dels grups on el teu usuari està.
b) Fes sortir per pantalla a quants grups estàs.
c) Fes sortir per pantalla les línies dels grups on estàs tu i algun usuari més.
d) Fes ara que només surtin els noms dels grups on està el teu usuari.
e) Emmagatzema aquests noms de grups en una variable que es digui MEUSGRUPS
f) Fes sortir per pantalla el primer i tercer camp (nom i identificador) de la línia corresponent al teu usuari a l'arxiu /etc/passwd. El nom i identificador estaran separats per ":".
g) Fes sortir el mateix resultat que a l'apartat f) però ara el nom i l'identificador estaran separats per un espai en blanc.
h) Assigna la sortida anterior a una variable anomenada IDENTITAT.
i) Crea una variable que es digui JO amb el contingut de la variable IDENTITAT i de la variable MEUSGRUPS, separats amb un $. Mostra el resultat per pantalla.

27. Volem imprimir per pantalla "Hola <num_id>" on num_id és el teu identificador. Fes-ho de tres maneres diferents:
a) Usant la comanda id amb opcions
b) Usant la comanda id sense opcions
c) Usant l'arxiu /etc/passwd

28. Usant la comanda date sense cap opció fes que:
a) Surti el dia (núm) per pantalla. Emmagatzemar el mes a la variable DIA.
b) Surti el mes per pantalla. Emmagatzemar el mes a la variable MES.
c) Surti l'any per pantalla. Emmagatzemar el mes a la variable ANY.
d) Surti per pantalla "Avui és dia <num_dia> del mes <mes> de l'any <any>", usant les variables creades.
e) Guardar la línia anterior en el fitxer avui.dat sense que la sortida es vegi per pantalla.
f) Guardar la línia anterior en el fitxer avui.dat i veient-se la sortida per pantalla.

:::SHELL-SCRIPTS:::

29. Dissenyar un script que donats dos arguments els sumi si el primer és menor que el segon i els resti en cas contrari.
30. Dissenyar un script que demani un caràcter i ens digui si és un número, una lletra o una altra cosa.
31. Dissenyar un script al qual se li passa un argument i si aquest és un directori llista el seu contingut.
32. Dissenyar un script que suma tots els números que se li passen per paràmetre.
33. Dissenyar un script que compti el número de caràcters que tenen els noms dels fitxers del directori actual.

34. Dissenyar un script que determini si els números que se li passen com a paràmetres són parells o senars.
35. Dissenyar un script que demani un nom d'usuari i ens digui si aquest existeix i, si existeix, ens digui si està conectat.
36. Dissenyar un script anomenat “lse” que et mostri per pantalla els fitxers i directoris que hi ha al directori actual de la següent manera:
Si amb la comanda ls obtenim la següent sortida

```
fit1   prog1   prog2   joc
```

Amb el nostre script lse ens ho mostrarà així:

```
|_fit1
.|_prog1
..|_prog2
...|_joc
```

37. Dissenyar un script anomenat “sumatam” que sumi les mides de tots els fitxers que se li passin com a arguments donant un error per a tots aquells arguments que no existeixin o que siguin directoris.
38. Dissenyar un script anomenat “sumadir” que sumi les mides de tots els fitxers del directori passat com a argument, donant un missatge d'error si l'argument passat no és un directori o no existeix.
39. Dissenyar un script anomenat “opf” que permeti copiar (-c), moure (-m) i esborrar (-d) fitxers. L'script ha de comprovar que la sintaxi utilitzada és la correcta.
40. Dissenyar un script anomenat “usua” que ens vagi demanant noms d'usuari i si aquests existeixen i estan connectats ens dirà la data i hora de la seva connexió. L'script s'acabarà quan l'usuari teclegi FI.
41. Dissenyar un script anomenat “lro” que mostri tots els fitxers amb permís de lectura per a altres d'un directori que es proporciona com a paràmetre.
42. Dissenyar un script anomenat “lus” que, donat un identificador de grup, llisti, per a cada usuari que pertany al grup, el seu id, el seu nom i el seu directori de treball. La sortida seria:

```
USUARIS DEL GRUP 52 (5)
NUM. ID.      NOM             DIR. TREBALL
-------------------------------------------------------------
235           maria           /home/maria
246           jordi           /home/jordi
...
```

43. Edita un fitxer anomenat “aniversari.txt” les línies del qual tinguin el següent format

```
nom:dataaniversari
```

Realitza un script anomenat “generaaniv” que et generi el fitxer que hauríem de passar a la comanda crontab per tal que el dia abans de cada data d'aniversari rebis un mail que tingui la següent informació:

```
...
Subject: Recordatori Aniversari
...
Demà dia [data] és l'aniversari de [nom]. No oblidis
felicitar-l@.
```

44. Dissenya crontabs perquè es realitzin les següents tasques:

- Cada dia a les 7 del matí miri quina mida té el directori /tmp i si és més gran a 10Mb l'esborri i envii un mail a l'administrador informant-lo
- Cada dos dies a les 12 de la nit, comprimeixi i empaqueti el directori /home i guardi aquest arxiu en el directori /usr/src/backups. Si el directori no existeix, es crearà.
- Cada dia, de dilluns a divendres, crei un informe a les 19:00, a les 20:00 i a les 21:00 amb els usuaris que estan connectats i l'envii a l'administrador. Aquest informe tindrà el següent aspecte:

```
DATA: [data]
HORA: [hora]

USUARIS CONECTATS ([numusuaris])
NUM. ID.      NOM     GRUP
-------------------------------------------------
[id1] [nomusu1]       [grupusu1]
[id2] [nomusu2]       [grupusu2]
[id3] [nomusu3]       [grupusu3]
...
```

:::EXPRESSIONS REGULARS:::

45. Crear les expressions regulars per obtenir cadenes que:
  a) continguin la cadena "aba"
  b) contenguin tres "b" seguides
  c) comencin per dos "a"
  d) acabin per "ba"
  e) comencin per "a" i acabin per "b" (al mig pot haver qualsevol cosa)
  f) només contingui "a" (la quantitat no importa)
  g) primer una "b" i després vàries "a"
  h) tingui tant "a" com "b" (l'ordre o la quantitat no importa)
  i) no tingui més de tres "a" o tres "b" seguides
  j) vagi alternant les "a" i les "b" sense repetir-se
  k) només tingui parelles de "a" i de "b"
  l) tinguin una única "a" o una única "b"
  m) hi hagi unes quantes "a" i després unes quantes "b" o al revés
  n) contengui la cadena "aba" o la cadena "bab"
  o) contingui la cadena "ba" dos cops

46. Crear les expressions regulars per obtenir cadenes que continguin:
  a) números decimals (amb una coma com a separador decimal)
  b) números decimals (con una coma o un punt com a separador decimal)
  c) números de telèfons (de nou xifres, que comencin per 9 o 6)
  d) codis postals (de cinc xifres, que comencin com a molt per 5)
  e) DNI (set o vuit xifres que poden anar seguides d'una lletra)

f) paraules en minúscules sense números
g) paraules en las quals només la primera lletra sigui majúscula
h) tres o quatre paraules (sense números)

47. Usant el fitxer /etc/passwd donar les expressions regulars per obtenir:
a) usuaris amb la paraula "Unix" al principi del camp de comentari.
b) usuaris del grup 101.
c) usuaris dels grups 100, 101 o 105.
d) llistar usuaris amb UID d'un dígit.
f) llistar usuaris amb UID de 1 o 2 dígits.
g) usuaris amb nom d'exactament 4 caràcters.
h) usuaris amb nom de 4 caràcters començant per r.

48. Fer expressions regulars per a:
a) Data (DD/MM/AAAA)
b) Hora
c) Correu electrònic
d) Números de telèfon: (93) 841.61.00
e) url's de tipus .org

4.2. links recomanats

:::shell scripts:::

http://www.tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf

Curs introductori de shell-scripts, força complert. Completaria opcions i alguns aspectes no explicats en aquest mini-curs

http://www.ciberdroide.com/misc/novato/curso/

Curs de GNU/Linux per consola. Té una part de shell-scripting.

:::expressions regulars:::

http://bulma.net/body.phtml?nIdNoticia=770

Tutorial d'expressions regulars.

http://iie.fing.edu.uy/~vagonbar/unixbas/expreg.htm

Tutorial d'expressions regulars.

http://gmckinney.info/resources/regex.pdf

Referència ràpida d'expressions regulars

:::awk:::

http://www.wikilearning.com/awk_paso_a_paso-wkc-31.htm

Petit tutorial de awk

http://www.inicia.es/de/chube/Manual_Awk/Manual_Awk_castellano.pdf

Tutorial complet de awk en castellà

4.3. off-topic

En cap moment no es pot dir *per sempre*. En canvi, sempre hi ha un moment en què cal dir *mai més*. Però: no és el mai més un per sempre? Tanmateix, el per sempre se situa en el temps, mentre el mai més és intemporal.

(Manuel de Pedrolo)