

MÒDUL

Sistemes Informàtics

UNITAT 5

Administració de GNU/Linux

ÍNDEX

1. Introducció
2. Ordres bàsics de Linux
3. Ordres de comptes d'usuari
4. Ordres d'instal·lació de programes
5. Ordres de gestió de processos
6. Ordre tar, rsync i crontab
7. Ordres de processament de text
8. Scripts

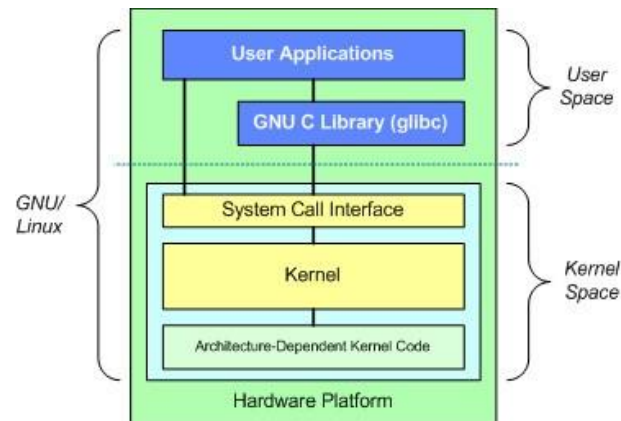
1. Introducció

Els sistemes GNU/Linux tenen una àmplia varietat d'usuaris i àmbits de treball on són utilitzats. El seu origen es remunta al mes d'agost del 1991, quan un estudiant finlandès anomenat **Linus Torvalds** va anunciar en una llista de news que hi havia creat el seu propi nucli de sistema operatiu i ho oferia a la comunitat de desenvolupadors perquè ho provés i suggerís millores per fer-ho més utilitzable. Aquest seria l'origen del **nucli** (o **kernel**) del sistema operatiu que més tard es diria **Linux**.

D'altra banda, la **FSF** (Free Software Foundation), mitjançant el seu **projecte GNU**, produïa software (des de 1984) que podia ser utilitzat lliurement. Degut a ho que **Richard Stallman** (membre de la FSF) considerava software lliure, és a dir, com aquell del que podíem aconseguir les seves fonts (codi), estudiar-lo, modificar-lo i redistribuir-lo sense que ens obliguin a pagar per això. En aquest model, el negoci no està en l'ocultació del codi, sinó al software complementari afegit, a l'adequació del software als clients i als serveis extres, com el manteniment i la formació d'usuaris (el suport que els donem), ja sigui en forma de material, llibres i manuals, o en cursos de formació.

UD5. ADMINISTRACIÓ DE GNU/LINUX

La combinació (o suma) del **software GNU** i del **kernel Linux**, és el que ens ha portat als actuals **sistemes GNU/Linux**. Actualment, els moviments OpenSource, des de diferents organitzacions (com FSF) i empreses com les que generen les diferents distribucions Linux (Xarxa Hat, Mandriva, SuSe, Ubuntu...), passant per grans empreses com HP, IBM o Sun que proporcionen suport, han donat una empenta molt gran als sistemes GNU/Linux fins a situar-los al nivell de poder competir, i superar, moltes de les solucions propietàries tancades existents.



UD5. ADMINISTRACIÓ DE GNU/LINUX

Hi ha més de 1600 distribucions de GNU/Linux diferent. Els Kernel de totes les distribucions de Linux comparteixen la mateixa línia evolutiva però cada distribució té els seus rutines de instal·lació, gestor de paquets, documentació específica, i una comunitat de suport. On elles difereixen essencialment és a la seva visió i característiques particulars per assolir tasques específiques. Algunes de les distribucions principals són

- **Debian:** Desenvolupada per voluntaris i enfocada a la utilització de software veritablement OpenSource, fàcil manteniment dels paquets i actualització del sistema.
- **Ubuntu:** Versió depurada de Debian. El projecte Ubuntu està patrocinat per Canonical Ltd, una companyia de holding fundada per el sud-africà Mark Shuttleworth.
- **Red Hat Enterprise:** Versió Red Hat Linux per a empreses, la seva venda és manejada per una entitat comercial focalitzada a suport i compatibilitat.
- **CentOS:** Versió gratuïta de Red Hat Enterprise.
- **Fedora:** Versió de Red Hat Linux no corporativa.

UD5. ADMINISTRACIÓ DE GNU/LINUX

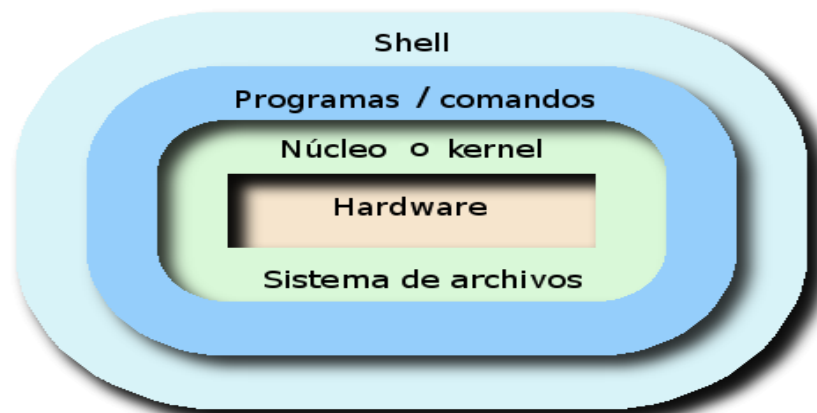
- **Slackware:** Una de les primeres distribucions; coneguda pel poc espai que utilitza del disc dur.
- **Gentoo:** Una distribució basada a codi font, coneguda per ser una distribució que és totalment compilada paquet a paquet.
- **SuSE:** Distribució Alemanya-Europea basada en paquets RPM amb orientació comercial i molt bon suport, i versions per a una gran varietat d'arquitectures.
- **Turbolinux:** Distribució de gran acollida a Àsia basada en RPM, acompanyada de solucions propietàries de high-end clustering (Nosaltres de Super Computadors) a més de distribuir versions de Servers i Estació de Treball (Workstation).
- **LinuxMint:** LinuxMint és una distribució del sistema operatiu GNU/Linux, basada a la distribució Ubuntu, també hi ha una versió basada en Debian. LinuxMint ve amb la seva propi joc de aplicacions (Mint tools) amb el objectiu de fer més senzilla la experiència del usuari.

UD5. ADMINISTRACIÓ DE GNU/LINUX



2. Estructura de GNU/Linux

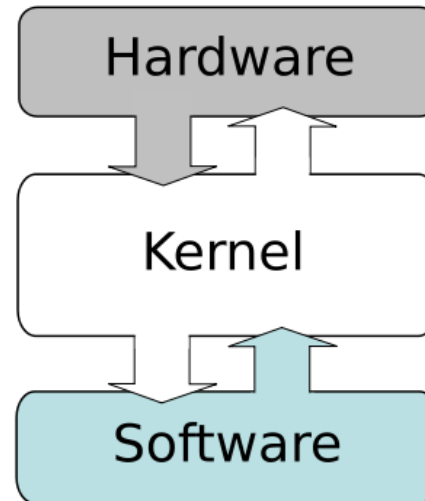
Per a comprendre el funcionament del sistema GNU/Linux cal comprendre la seva estructura. Aquest sistema operatiu està format per diversos components principals. Entre ells, el **Nucli** (oKernel), el **Shell**, el **sistema de fitxers**, els **programes** i els **ordres**.



El kernel

El **Kernel** és la part del sistema operatiu que serveix per a **interactuar amb el hardware**. Proporciona una sèrie de serveis que poden ser utilitzats pels programes, sense que aquests tinguin que preocupar-se de com es gestiona el hardware.

En general, el **nucli** és l'encarregat de gestionar la memòria, mantenir el sistema de arxius, del maneig d'interrupcions, maneig d'errors, realització dels serveis de entrada/sortida, assignació de els recursos de la CPU, comunicació entre processos, etc.



UD5. ADMINISTRACIÓ DE GNU/LINUX

El kernel es carrega en memòria quan el sistema s'inicia i **roman en memòria** fins que el sistema es descarrega del tot. Es dissenya per ser el més petit que sigui possible, permetent així que la memòria restant sigui compartida entre tots els programes que s'executen al sistema. Els programes es relacionen amb l'ordinador a través del nucli mitjançant les **trucades al sistema**.

El Kernel sol estar escrit a **C**, amb una mica de llenguatge **assemblador** que li facilita treballar directament amb el hardware. Un dels avantatges dels sistemes Linux és que, al estar el codi font del kernel disponible, és més senzill desenvolupar els nostres propis controladors i mòduls del kernel.

El Shell

El **Shell** és la part que permet a l'usuari comunicar-se amb el sistema. Es pot estudiar el Shell des de dos punts de vista: com **intèrpret d'ordres** i com **llenguatge de programació**, que combina mitjançant estructures de control grups de ordres emmagatzemats a arxius anomenats **Shell Scripts** o procediments Shell.

Quan l'usuari introdueix una ordre, el Shell, que és un programa continuat execució, analitza la línia i crida al programa o programes que realitza la funció sol·licitada per l'ordre.

L'administrador del sistema deu adjudicar un shell a cada usuari. Aquest, a a l'hora, podeu executar qualsevol Shell sempre que tingui autorització per fer-ho. Quan un usuari es connecta al sistema s'inicia automàticament un programa de shell denominat **Shell de presentació**. Aquest Shell es carrega de forma automàtica quan s'accedeix al fitxer **/etc/passwd**, que conté informació dels usuaris inclòs el shell estàndard que utilitzarà cada un.

UD5. ADMINISTRACIÓ DE GNU/LINUX

El sistema de fitxers

Linux defineix una interfície abstracta al nivell del kernel que incorpora diversos sistemes de arxius diferents.



UD5. ADMINISTRACIÓ DE GNU/LINUX

El contingut de alguns de els directoris més importants es resumeix a la següent taula:

Ruta	Contingut
/	Directorí arrel del sistema
/bin	Ordres i programes essencials
/boot	Kernel i arxius necessaris per carregar-lo (grub2)
/dev	Tots els dispositius físics del sistema (discos, impressores..)
/etc	Fitxers de configuració e inici
/home	Directorí principal de els usuaris
/lib	Lliberies i parts del compilador de C
/media	Unitats físiques muntades (discos durs, DVDs, pen drives..)
/mnt	Sistema de arxius muntats temporalment
/opt	Paquets de programari per aplicacions opcionals
/proc	Informació sobre els processos carregats
/root	Directorí principal del superusuari
/sbin	Ordres per iniciar, reparar i recuperar el sistema
/srv	Dades servides pel sistema
/sys	Àrea de treball del kernel i arxius de configuració
/tmp	Fitxers temporals
/usr	Dades d'usuari, conté la majoria de les utilitats i aplicacions
/var	Fitxers variables, com a logs i temporals

2. Ordres bàsiques de Linux

ls

És la primera ordre que tot linuxer ha d'aprendre. Ens mostra el contingut de la carpeta que us indiquem després. Per exemple. Si volem que ens mostri el que conté /etc:

```
ls /etc
```

Si no hi posem res interpretarà que el que volem veure és el contingut de la carpeta on estem actualment.

A més, accepta certs arguments que poden ser interessants. Per mostrar tots els fitxers i carpetes, incloent-hi els ocults:

```
ls -a
```

Per mostrar els fitxers i carpetes juntament amb els drets que té, el que ocupa, etc:

```
ls -l
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

A més, es poden solapar els arguments. Si volguéssim mostrar els arxius de la mateixa manera que abans, però que mostri també els ocults:

`ls -la`

Altres paràmetres per `ls` són:

- h: mostra la memòria que ocupa cada fitxer i el que ocupa el contingut de cada memòria.
- r: mostra el llistat en ordre invers (de la z a la a).
- S: mostra el llistat ordenat per mida (de major a menor).
- t: mostra el llistat ordenat per temps (de més recent a més antic).
- R: mostra també el contingut dels subdirectoris.

cd

Change directory: canviar directori. Podem fer-lo servir amb rutes absolutes o relatives. A les absolutes us indiquem tota la ruta des de l'arrel (/). Per exemple, estiguem on estiguem, si escrivim en consola...

```
cd /etc/apt
```

... ens portarà a aquest directori directament. De la mateixa manera si escrivim

```
cd ..
```

ens portarà al directori pare de la nostra ubicació actual.

```
cd /
```

ens enviarà a l'arrel del sistema de fitxers.

UD5. ADMINISTRACIÓ DE GNU/LINUX

Les rutes relatives són relatives a alguna cosa, i aquesta cosa és el directori on estiguem actualment. Imagineu que estem a /home i volem anar a un directori que es diu temporal dins de la vostra carpeta personal. Amb escriure...

```
cd la teva_carpeta/temporal
```

... ens situarà allà. Com veieu hem obviat el /home inicial ja que si no ho introduïm pren com a referència el directori on estem, que és aquest.

I què passa si escrivim només...

```
cd
```

Això el que fa és que et porti al teu directori personal directament estiguem on estiguem.

```
cd -
```

Ens torna l'últim directori on estiguéssim.

mkdir

Make directory: Crea un directori amb el nom que us indiquem. Novament podem fer servir rutes absolutes i relatives. Podem indicar-li tota la ruta que precedeix al directori que volem crear, o si estem ja a la carpeta que el contindrà només cal posar només el nom:

```
mkdir /home/el teu_compte/cogombre
```

si ja estem a /home/el teu_compte...

```
mkdir cogombre
```

nano

Editor de fitxers de text

rm

remove: esborrar. Esborra el fitxer o la carpeta que us indiquem. Com abans es pot indicar la ruta completa o el nom del fitxer. Això a partir d'ara ho obviarem, crec que ja ha quedat clar amb les dues ordres anteriors.

Per esborrar un fitxer:

```
rm nom_arxiu
```

Per esborrar una carpeta buida:

```
rm nom_carpeta
```

Per esborrar una carpeta que conté fitxers i/o altres carpetes que poden fins i tot contenir més:

```
rm -r nom_carpeta
```

Altres opcions: “-f” no us demana una confirmació per eliminar o “-v” va mostrant el que va esborrant. -i: sí que et demana confirmació.

UD5. ADMINISTRACIÓ DE GNU/LINUX

cp

copy: copiar. Copia el fitxer indicat on li diguem. Aquí també podem jugar amb les rutes, tant per al fitxer origen, com en el de la destinació. També podeu posar el nom que voleu posar a la còpia. Per exemple, si estiguéssim a /etc/X11 i voldríem fer una còpia de seguretat de xorg.conf a la nostra carpeta personal:

```
cp xorg.conf /home/la_teva_carpeta/xorg.conf.backup
```

Paràmetres:

-i: perquè no sobrescrigui xicots. (avisa si es vol sobreescriure)

-b: perquè en lloc de sobreescriure arxius creï un amb el mateix nom afegint-li un ~

Per copiar un directori es fa servir `cp -r`

```
cp -r /etc /backup/
```

Tindrem /backup/etc, i dins d'aquest directori tindrem el mateix que hi ha a /etc.

UD5. ADMINISTRACIÓ DE GNU/LINUX

mv

move: moure. És igual que l'anterior, només que en lloc de fer-ne una còpia, mou directament el fitxer amb el nom que li indiquem, pot ser un altre diferent de l'original:

```
mv /etc/cogombre.html /home/la teva_carpeta/aquest_pepino.html
```

Un altre ús molt pràctic que se li pot donar és per reanomenar un fitxer. Només cal indicar el nou nom al segon argument amb la mateixa ruta del primer. En aquest exemple suposem que ja estem a la carpeta que el conté:

```
mv cogombre.html ese_cogombre.html
```

Paràmetres:

- i: perquè no sobrescrigui arxius. (avisa si es vol sobre escriure)
- b: perquè en lloc de sobre escriure arxius creï un amb el mateix nom afegint-li un ~

UD5. ADMINISTRACIÓ DE GNU/LINUX

find

find: trobar. Busca el fitxer o carpeta que li indiquis:

```
find / -name cogombre
```

La comanda anterior buscaria a tot arreu les carpetes i arxius que s'anomenin cogombre. Si tinguéssim la seguretat que es troba a /var per exemple, els ho indicariem:

```
find /var -name cogombre
```

Si no estem gaire segurs del nom podem indicar-ho amb comodins. Suposem que el nom del que busquem conté “pepi”, a la mateixa carpeta d'abans:

```
find /var -name *pepi*
```

Teniu altres opcions. Per exemple us podem dir que trobeu els fitxers/carpetes de més de 1500 KB:

```
find / -size +1500
```

O els fitxers/carpetes contenen el nom “pepi” i tenen menys de 1000 KB:

```
find / -name *pepi* -size -1000
```

clear

clear: buidar. Netegeu la pantalla/consola quedant-la com si acabéssim d'obrir-la. En el fons no esborra tot, fa que elprompt pugueu a la primera línia, però si feu scroll cap amunt podreu veure el que teníeu abans.

clear

man

manual: manual. És un altre dels comandaments de gran potència alinux. Normalment cada programa o ordre ve amb un fitxer d'ajuda complet sobre el seu ús i els seus arguments. Quan desconegueu com s'usa i quins arguments té una ordre o aplicació tan sols heu d'escriure a la consola:

man nom

De vegades la informació que ens ofereix manpot ser excessiva. Gairebé totes les ordres i aplicacions accepten l'argument "--help"perquè mostra certa ajuda més resumida.

UD5. ADMINISTRACIÓ DE GNU/LINUX

pwd

Mostra el directori on estem situats.

cat

Mostra per pantalla contingut d'un fitxer

pwd

Mostra el directori on estem situats.

more

Canonada per pausar la sortida d'una ordre a cada canvi de pantalla.

Redireccionaments:

>: Redirecciona la sortida d'una ordre a un fitxer.

>>: Annexa la sortida d'una ordre al contingut d'un fitxer.

UD5. ADMINISTRACIÓ DE GNU/LINUX

alias

Permet crear dreceres per a ordres:

```
alias ll="ls -s"
```

du

Indica l'espai que ocupa un directori

df

Mostra l'espai de disc usat als sistemes de fitxers de totes les particions

```
df -h
```

free

Mostra la quantitat de memòria lliure i usada al sistema

```
free -m
```

3. Comandes de comptes d'usuari

Afegir un usuari

```
suo adduser <nom_d'usuari>
```

Eliminar un usuari

```
sudo deluser <usuari>  
sudo userdel <usuari>
```

Suprimiu un usuari i la vostra carpeta personal

```
sudo userdel -r <usuari>
```

Bloquejar/desbloquejar un compte d'usuari

```
sudo passwd -l <usuari>  
sudo passwd -u <usuari>
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Assignar una contrasenya a un usuari

```
sudo passwd <usuari>
```

Crear un grup d'usuaris

```
sudo addgroup <nomgrup>
```

Afegir un usuari a un grup

```
sudo adduser <usuari> <grup>
```

Treure un usuari d'un grup

```
sudo deluser <usuari> <grup>
```

Suprimir un grup d'usuaris

```
sudo delgroup <grup>
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Canvia el nom d'un usuari

```
sudo usermod -l <nounom> <nomactual>
```

Canvia el nom d'un grup d'usuaris

```
sudo groupmod -n <nounom> <nomactual>
```

Fitxer que conté la informació dels usuaris.

/etc/passwd

Fitxer que conté la informació dels grups d'usuaris.

/etc/group

UD5. ADMINISTRACIÓ DE GNU/LINUX

Mostra els grups a què pertany l'usuari actual

groups

Obrir una sessió de terminal d'un altre usuari

sudo el seu <usuari>

Mostrar l'id de l'usuari i el dels grups a què pertany

id

Permisos d'usuaris

Canviar l'usuari propietari d'un fitxer.

```
sudo chown <usuari> <fitxer>
```

Canviar el grup a què pertany un fitxer.

```
sudo chgrp <grup> <fitxer>
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Canviar els permisos d'un fitxer

`sudo chmod <mode> nom del fitxer`

On un fitxer té tres grups d'atributs de fitxers de permisos de lectura, escriptura i execució:

`rwX rwX rwX`

El primer grup pertany als drets del propietari del fitxer.

El segon grup pertany al grup propietari del fitxer.

El tercer grup pertany a la resta d'usuaris.

Aquests permisos s'estableixen amb tres valors de 0 a 7, és a dir, cada valor correspon a un grup i la seva traducció al binari fa que en el permís on hi hagi un 1 s'activa el permís i on hi hagi un 0 es desactiva el permís.

Per exemple:

`sudo chmod 750`, es traduiria `111 101 000`

per tant activaria els següents permisos `rwX rx ---`

UD5. ADMINISTRACIÓ DE GNU/LINUX

Una altra forma de treballar és usant caràcters on es creen diferents grups que se'ls identifica per un caràcter:

u: usuari propietari

g: grup propietari

o: altres

a: tots

Usant els caràcters + o - se'ls afegeix o treu als fitxers els permisos especificats, per exemple:

```
sudo chmod go+w fitxer
```

```
sudo chmod ow fitxer
```


ACLs

Amb els comandaments anteriors, els permisos s'estableixen al usuari propietari d'un arxiu, al grup propietari de l'arxiu o al altres usuaris, pero si volem donar permisos específics a usuaris o grups d'usuaris que no estiguen a aques grup hem d'utilitzar les ACL que son regles on s'especifiquen permisos concrets.

Llistar les ACLs:

```
getfacl
```

Assignar una ACL:

```
setfacl -R -m [u|g|o]:[usuari|grup]:[r,w,x] <fitxer>
```

Eliminar una ACL:

```
setfacl -x user: <usuari> <fitxer>
```

4. Comandes d'instal·lació de programes

Instal·lar un programa

```
sudo apt-get install paquet
```

Actualitzar la llista de paquets disponibles

```
sudo apt-get update
```

Actualitzar el sistema

```
sudo apt-get upgrade
```

Desinstal·lar un programa

```
sudo apt-get remove paquet
```

5. Comandes de gestió de processos

Mostra els processos dusuari llançats.

ps

Mostrar tots els processos

ps aux

Matar un procés.

kill <id_del_procés>

Matar un procés de manera més robusta

kill -9 <id_del_procés>

UD5. ADMINISTRACIÓ DE GNU/LINUX

Pausar un procés

```
kill -STOP <id_del_procés>
```

Reprendre un procés.

```
kill -CONT <id_del_procés>
```

Matar un procés prement sobre la finestra.

```
xkill
```

6. Ordres tar, rsync i crontab

tar

L'ordre tar s'utilitza per agrupar diversos fitxers o directoris en un mateix contenidor, també tenim l'opció de comprimir aquest contenidor.

Ta r[opcions] “contenidor” “fitxers1 fitxer2 ...fitxerN”

Opcions:

- c: Crea el contenidor. S'usa sempre que el contenidor no existeixi.
- v: Llista per pantalla els fitxers que va afegint al contenidor.
- f: Quan volem fer referència a un contenidor, per exemple per extreure'n certs documents o per afegir-los a aquest arxivador (contenidor).
- z: Si volem que el contenidor estigui comprimit. Un fitxer comprimit ocupa menys espai físic.

UD5. ADMINISTRACIÓ DE GNU/LINUX

- x: Opció utilitzada per a descomprimir un contenidor comprimit.
- t: Mostra tots els fitxers i directoris d'un contenidor. Aquest contenidor ha d'estar descomprimit.
- r: Afegeix un fitxer/directori a un contenidor.
- d: Indica les diferències entre els fitxers/directoris que conté el contenidor i els fitxers/directoris existents al directori actual (pwd).
- delete: Elimina un fitxer del contenidor. Només funciona si NO està comprimit.
- u: Actualitza el contenidor. És a dir, si un fitxer es troba tant al contenidor com al directori i la data de modificació és més recent a la carpeta, l'actualitza al contenidor. Només funciona si NO està comprimit.

UD5. ADMINISTRACIÓ DE GNU/LINUX

Els contenidors o arxivadors usen l'extensió.tar.

```
-rw-rw-r-- 1 miguel miguel 20480 mar 19 12:43 miContenedor.tar
```

Els contenidors o arxivadors comprimits usen l'extensió.tgz.

```
-rw-rw-r-- 1 miguel miguel 123 mar 19 12:45 miContenedor.tgz
```

És moltIMPORTANT l'ús del "*" per indicar tots els fitxers i directoris de la carpeta actual.

UD5. ADMINISTRACIÓ DE GNU/LINUX

Exemples:

Tenim la següent carpeta amb els següents fitxers/directoris:

```
miguel@miguel-VirtualBox:~/prueba$ ls
alumnos  aprobados.txt  notas.txt  suspensos.txt
```

1. Creem un contenidor (arxivador) que contingui els fitxers “notes.txt” “suspensos.txt” i la carpeta “alumnes”. El contenidor l'anomenarem “exemple1.tar”.

```
miguel@miguel-VirtualBox:~/prueba$ tar -cvf ejemplo1.tar notas.txt suspensos.txt alumnos/alumnos.txt suspensos.txt
notas.txt
suspensos.txt
alumnos/alumnos.txt
suspensos.txt
```

Com es pot comprovar em crea un contenidor anomenat “exemple1.tar”. Com que fem servir l'opció -v ens surten llistats tots els fitxers que s'han anat inserint a l'arxivador.

UD5. ADMINISTRACIÓ DE GNU/LINUX

2. Farem servir l'opció `-t` per a visualitzar que hi ha dins del contenidor. És IMPRESCINDIBLE combinar-ho amb l'opció `-f` per a indicar el contenidor que llistarem.

```
miguel@miguel-VirtualBox:~/prueba$ tar -tf ejemplo1.tar
notas.txt
suspensos.txt
alumnos/alumnos.txt
suspensos.txt
```

3. Ara afegirem el fitxer aprovats.txt al contenidor. Per fer-ho farem servir l'opció `-r`.

```
miguel@miguel-VirtualBox:~/prueba$ tar -rf ejemplo1.tar aprobados.txt
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

4. Eliminem tots els fitxers de la carpeta (notes, aprovats, suspensos i alumnes) i descomprimim tots els fitxers en aquesta carpeta. Fixa't com hem fet servir l'opció “x”, de nou IMPRESCINDIBLE que vagi associat a l'opció –f per poder fer referència a quin arxivador.

```
miguel@miguel-VirtualBox:~/prueba$ rm -rf notas.txt suspensos.txt alumnos/alumnos.txt suspensos.txt aprobados.txt
miguel@miguel-VirtualBox:~/prueba$ ls
alumnos  ejemplo1.tar
miguel@miguel-VirtualBox:~/prueba$ tar -xf ejemplo1.tar
miguel@miguel-VirtualBox:~/prueba$ ls
alumnos  aprobados.txt  ejemplo1.tar  notas.txt  suspensos.txt
```

5. Ara tornem a eliminar tots els fitxers, però aquesta vegada només descomprimim un fitxer del contenidor, per exemple, notes.txt.

```
miguel@miguel-VirtualBox:~/prueba$ rm -rf notas.txt suspensos.txt alumnos/alumnos.txt suspensos.txt aprobados.txt
miguel@miguel-VirtualBox:~/prueba$ tar -xf ejemplo1.tar notas.txt
miguel@miguel-VirtualBox:~/prueba$ ls
alumnos  ejemplo1.tar  notas.txt
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

6- Ara, usant l'opció `-d`, veurem les diferències entre el que hi ha al meu contenidor i al directori actual. Recordeu que actualment al directori només hi ha el fitxer `notes.txt`.

```
miguel@miguel-VirtualBox:~/prueba$ tar -df ejemplo1.tar
tar: suspensos.txt: Atención: No se puede stat: No existe el archivo o el directorio
tar: alumnos/alumnos.txt: Atención: No se puede stat: No existe el archivo o el directorio
tar: suspensos.txt: Atención: No se puede stat: No existe el archivo o el directorio
tar: aprobados.txt: Atención: No se puede stat: No existe el archivo o el directorio
```

7-Eliminem la carpeta `alumnos` del contenidor. Posteriorment fem servir l'opció `-t` per confirmar que efectivament ha estat eliminada aquesta carpeta del contenidor.

```
miguel@miguel-VirtualBox:~/prueba$ tar --delete -f ejemplo1.tar alumnos/
miguel@miguel-VirtualBox:~/prueba$ tar -t -f ejemplo1.tar
notas.txt
suspensos.txt
suspensos.txt
aprobados.txt
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

8. Veurem un exemple d'actualització dels fitxers.

a. Eliminem tots els fitxers: (notes.txt, perquè suspensos.txt i aprovats.txt.

```
miguel@miguel-VirtualBox:~/prueba$ rm notas.txt
```

b. Descomprimeixo únicament el fitxer notes.txt

```
miguel@miguel-VirtualBox:~/prueba$ tar -xf ejemplo1.tar notas.txt  
miguel@miguel-VirtualBox:~/prueba$ ls  
ejemplo1.tar  notas.txt
```

c. Modifico el fitxer notes.txt amb el contingut que vulgui (usa l'editor nano).

d. Actualitzo el contenidor. Fixa't en l'ús del * per fer referència a tots els fitxers que trobi dins la carpeta actual. Fem servir l'opció -v perquè ens informe dels fitxers que heu actualitzat.

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
miguel@miguel-VirtualBox:~/prueba$ tar -uvf ejemplo1.tar *  
tar: ejemplo1.tar: el fichero es el propio archivo; no se vuelca  
notas.txt
```

e. Esborra notes.txt de la carpeta.

f. Descomprimeixo el notes.txt del contenidor i comprovo que efectivament està actualitzat.

9. Exemple de creació d'un contenidor comprimit. De nou fem ús del * per fer referència a tots els fitxers i directoris de la meva carpeta. Fixa't que fem servir l'opció -c perquè

```
miguel@miguel-VirtualBox:~/prueba$ tar -czvf miContenedor.tgz *  
ejemplo1.tar  
notas.txt
```

10. També podem niar ordres amb l'ordretar. El següent exemple crea un contenidor amb tots els fitxers que han estat modificats el darrer dia.

```
tar -cfcontenedor.tar `find . -mtime-1 -type f`
```

rsync

- Ordre que permet fer còpies locals o remotes
- Realitza còpies incrementals
- Sintaxi: `rsync [opcions] origen destí`
- Les opcions més importants són:
 - ❑ `-v`: mode verbose
 - ❑ `-a`: es fa sobre arxius
 - ❑ `-h`: per treure la informació en múltiples de mesura (kB, MB,...)
 - ❑ `--delete`: fa que si s'ha esborrat un fitxer a l'origen s'esborri també de la còpia

rsync

```
miguel@miguel-VirtualBox:~$ mkdir copia
miguel@miguel-VirtualBox:~$ rsync -avh /home/miguel/prueba /home/miguel/copia
sending incremental file list
prueba/
prueba/ejemplo1.tar
prueba/miContenedor.tgz
prueba/notas.txt

sent 62.08K bytes  received 77 bytes  124.31K bytes/sec
total size is 61.78K  speedup is 0.99
```

```
miguel@miguel-VirtualBox:~$ cd prueba
miguel@miguel-VirtualBox:~/prueba$ nano notas.txt
miguel@miguel-VirtualBox:~/prueba$ rm ejemplo1.tar
miguel@miguel-VirtualBox:~/prueba$ cd ..
miguel@miguel-VirtualBox:~$ rsync -avh --delete /home/miguel/prueba /home/miguel/copia
sending incremental file list
deleting prueba/ejemplo1.tar
prueba/
prueba/notas.txt

sent 10.44K bytes  received 62 bytes  21.00K bytes/sec
total size is 10.58K  speedup is 1.01
miguel@miguel-VirtualBox:~$
```

crontab

- Es fa servir per automatitzar tasques a certes hores del dia.
- El dimoni “crond” és el que s'encarrega del funcionament correcte de crontab.
- S'executa en segon pla.
- Cada usuari, inclòs root, té el seu propi fitxer de crontab, les tasques dels quals s'executen encara que el vostre usuari no es trobi allà en aquell moment.
- El fitxer d'opcions s'emmagatzema a “/var/spool/cron/crontabs”, però només cal modificar-lo amb l'eina crontab.
- Els errors es registren a /var/log/syslog

```
crontab [-u user] file
```

```
crontab [-u user] [-l | -r | -e] [-i]
```


crontab

- Configuració:

1. Executem crontab

- a. crontab -e

- b. sudo crontab -e.

2. Accedirem al fitxer de configuració:

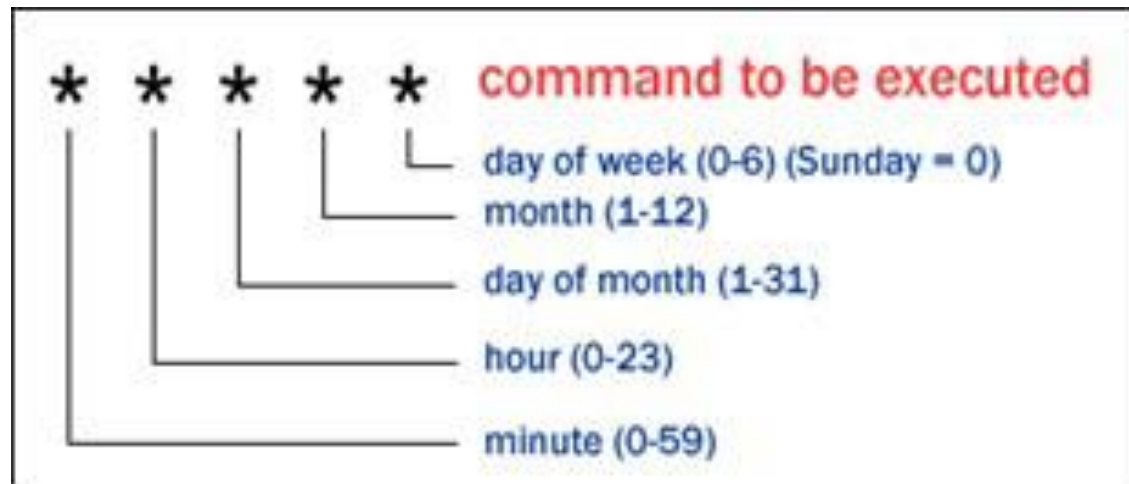
```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

crontab

crontab [-e | -l | -r] [usuari]

- -e: inicia l'edició del cron
- -l: veure les tasques programades al fitxer cron
- -r: esborrar un fitxer cron.

Síntaxi:



crontab

3. Cada línia que omplim del fitxer ha de tenir el patró següent:

- a. min hour day month weekday command
- b. (0-59) (0-23) (1-31) (1-12) (0-7) comanda
- c. Es pot fer ús del “*” per indicar que qualsevol valor complirà la premissa.
És a dir si a l'apartat de mes hi ha un asterisc, la tasca es farà tots els mesos.
- d. Es pot fer ús de la coma per indicar dos valors:
0 0,12 1 12 * comanda.
- e. Es pot fer ús del “-” per indicar una sèrie:
0 0-6 1 12 * comanda

Exemples de contrab

- Fes que es comprimeixi una carpeta creada per tu d'aquí a 5 minuts.

```
sad@sad-VirtualBox:~$ crontab -e
no crontab for sad - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/ed
 2. /bin/nano      <---- easiest
 3. /usr/bin/vim.tiny

Choose 1-3 [2]: 2
```

```
5 * * * * tar /home/sad/campus campus.tar
```

```
sad@sad-VirtualBox:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
5 * * * * tar /home/sad/campus campus.tar
```

- Fes que es comprimeixi una carpeta creada per tu tots els divendres a les 3 del matí
- Fes que es comprimeixi una carpeta creada per tu cada dilluns i divendres a les 2:30 del matí

7. Comandes de processament de text

sort

Ordena alfabèticament línies de text i les mostra a la sortida estàndard.

En la forma més simple, tot el que cal fer és donar-li el nom del fitxer a ordenar.

Algunes opcions:

- b ignora blancs al principi de línia*
- f no distingeix majúscules/minúscules*
- r ordre invers*
- m barreja fitxers prèviament ordenats*
- n ordena numèricament*
- k POS1[, POS2] ordena segons els camps des de POS1 a POS2, o el final si no hi ha POS2 (el primer camp és 1)*

UD5. ADMINISTRACIÓ DE GNU/LINUX

Exemples:

```
$ cat noms.txt
    Maria Pérez
    luis Andiό
    Adriana Gómez
    jorge pena
$ sort noms.txt
    Adriana Gómez
    Maria Pérez
    jorge pena
    luis Andiό
$ sort -f noms.txt
    Adriana Gómez
    jorge pena
    luis Andiό
    Maria Pérez
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
$ sort -f +1 +0 noms.txt #Obsolet (no fer servir)
    luis Andiό
    Adriana Gόmez
    jorge pena
    Maria Pέrez
```

```
$ sort -f -k 2,2 noms.txt
    luis Andiό
    Adriana Gόmez
    jorge pena
    Maria Pέrez
```


UD5. ADMINISTRACIÓ DE GNU/LINUX

La substitució d'un fitxer per una versió ordenada és tan freqüent que a UNIX hi ha una opció `-o` (output) que es pot utilitzar per dir a `sort` que substitueixi el fitxer d'entrada amb la versió ordenada. El següent ordena `fitxer1` i substitueix el seu contingut amb el contingut ordenat:

```
$ sort -o fitxer1 fitxer1_ordenat
```

Noteu que no es pot redirigir simplement la sortida de `sort` al fitxer original. Com que el shell crea el fitxer de sortida abans d'executar l'ordre, el que eliminaria el fitxer original.

Especificació del separador de camp `sort` permet especificar un separador de camp alternatiu. Això es pot fer amb l'opció `-t` (tab). L'ordre següent diu a `sort` que salti els primers tres camps d'un fitxer que utilitza dos punts (`:`) com a separador de camps:

```
$ sort -t: -k2 fitxer
```

Uniq

Aquest filtre suprimeix files duplicades d'un fitxer ordenat.

```
bash$ cat testfile  
This line occurs only onze.  
This line occurs twice.  
This line occurs twice.  
This line occurs three times.  
This line occurs three times.  
This line occurs three times.
```

```
bash$ uniq -c testfile
```

```
1 This line occurs only onze.  
2 This line occurs twice.  
3 This line occurs three times.
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
bash$ sort testfile | uniq -c | sort -nr
```

```
3 This line occurs three times.  
2 This line occurs twice.  
1 This line occurs only onze.
```

```
$ who | cut -f1 -d' '
```

```
root  
murie  
murie  
practica
```

El resultat no és del tot perfecte. Falta eliminar la doble aparició de murie. Dit i fet.

```
$ who | cut -f1 -d'' |uniq  
root  
murie  
practica
```

cut

Escriu parts seleccionades d'un fitxer a la sortida estàndard; es pot utilitzar per seleccionar columnes o camps d'un fitxer específic.

Algunes opcions:

-b, -c, -f talla per bytes, caràcters o camps, respectivament
-d fixa el caràcter delimitador entre camps (per defecte, TAB)

Exemples:

```
$ cut -f1,7 -d: /etc/passwd
```

```
root:/bin/bash
```

```
murie:/bin/bash
```

```
pràctica:/bin/ksh
```

```
wizard:/bin/bash
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
$ cat noms-ord.txt
    Luís Andi3n
    Adriana G3mez
    Jorge Pena
    Maria P3rez

$ cut -c 1-7 noms-ord.txt
    Lluís An
    Adriana
    Jorge P
    Maria P

$ cut -c 1-5,9-10 noms-ord.txt
    Lluís i3
    AdriaG3
    Jorgena
    Mariare

$ cut -d ' ' -f 1 noms-ord.txt
    Lluís
    Adriana
    Jordi
    Maria
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Especificació de delimitadors

S'utilitza el paràmetre “*d*” per a indicar quin és el caràcter separador de camps quan aquests no estiguin separats per un únic caràcter blanc.

El fitxer */etc/passwd* conté com a caràcter separador de camp els “:”. La següent ordre selecciona el nom de presentació, nom d'usuari i directori de presentació (camps 1, 5 i 6) del fitxer esmentat:

```
$ cut -d: -f 1,5-6 /etc/passwd
```

Si el delimitador té un significat especial per al *shell*, hauria d'anar entre cometes. Per exemple, la següent ordre diu a *cut* que imprimeixi tots els camps a partir del segon, utilitzant un espai com a delimitador.

```
cut -d' ' -f2- file
```

paste

Permet unir text de diversos fitxers, unint les línies de cadascun dels fitxers.

Algunes opcions:

- s enganxa els fitxers seqüencialment, en comptes d'intercalar-los
- d especifica els caràcters delimitadors a la sortida (per defecte, TAB)

```
bash$ cat ítems  
    alphabet blocks  
    building blocks  
    cables
```

```
bash$ cat prices  
    $1.00/dozen  
    $2.50 ea.  
    $3.75
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
bash$ paste ítems prices
    alphabet blocks $1.00/dozen
    building blocks $2.50 ea.
    cables $3.75
```

```
$ cat noms.txt
```

```
Lluís
Adriana
Jordi
Maria
```

```
$ cat cognoms.txt
```

```
Andió
Gómez
Pena
Pérez
```

```
$ paste noms.txt cognoms.txt
```

```
Luis Andión
Adriana Gómez
Jorge Pena
Maria Pérez
```


UD5. ADMINISTRACIÓ DE GNU/LINUX

```
$ paste -d ' ' noms.txt cognoms.txt
Luis Andión
Adriana Gómez
Jorge Pena
Maria Pérez
```

```
$ paste -s -d '\t\n' noms.txt
Luis Adriana
Jorge Maria
```

Especificació del separador de camp en paste

Paste separa les línies que uneix amb un separador de camp

El delimitador per defecte és el tabulador, però com passa amb *cut*, es pot utilitzar l'opció `-d` (delimitador) per especificar-ne un altre si es vol.

```
$ paste -d: file1 file2 file3
```

L'ordre anterior combina els fitxers d'estats utilitzant els dos punts com a delimitador

join

Permet combinar dos fitxers usant camps: cerca als fitxers per entrades comuns al camp i uneix les línies; els fitxers han d'estar ordenats pel camp d'unió (aquest apareix només una vegada a la sortida)

Per defecte, *join* utilitza el primer camp de cada fitxer d'entrada com a camp comú d'unió. Es poden utilitzar altres camps com a camp comú amb l'opció `-j` (*join*).

Algunes opcions:

- i ignora majúscules/minúscula*
- 1 FIELD uneix al camp FIELD (sencer positiu) de fitxer1*
- 2 FIELD uneix al camp FIELD de fitxer2*
- j FIELD equivalent a -1 FIELD -2 FIELD*
- t CHAR utilitza el caràcter CHAR com a separador de camps*
- o FMT formata la sortida (MN fitxer M camp N, O camp d'unió)*
- v N en comptes de la sortida normal, mostra les línies que no s'uneixen del fitxer N*
- a N a més la sortida normal, mostra les línies que no s'uneixen del fitxer N*

UD5. ADMINISTRACIÓ DE GNU/LINUX

Exemple:

```
$ cat noms1.txt
    Luis Andi3n
    Adriana G3mez
    Jorge Pena
    Maria P3rez

$ cat noms2.txt
    Pedro Andi3n
    Celia Fern3ndez
    Juliol Lorenzo
    Enric Pena

$ join -j 2 noms1.txt noms2.txt
    Andi3 Luis Pedro
    Pena Jorge Enrique

$ join -j 2 -o 1.1 2.1 0 noms1.txt noms2.txt
    Luis Pedro Andi3n
    Jorge Enrique Pena
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Especificació de separadors de camp

join tracta qualsevol espai en blanc (un espai o un tabulador) a l'entrada com a separador de camp. Utilitza el caràcter espai en blanc com a delimitador per defecte a la sortida.

Amb l'opció `-t` (tab) es pot canviar el separador de camp. L'ordre següent uneix les dades dels fitxers del sistema `/etc/passwd` i `/etc/group`, utilitzant tots dos punts com a separador de camp. El mateix separador s'utilitza per a l'entrada i la sortida.

```
$ join -t: /etc/passwd /etc/group > destinació
```

Malauradament, la lletra d'opció que utilitza *join* per especificar el delimitador (`-t`) és diferent de la (`-d`) utilitzada per *cut*, *pasta* i altres ordres del sistema UNIX.

head

Mostra el principi de fitxer.

Algunes opcions:

-n N o -N mostra les primeres N línies

-c N mostra els primers n bytes

-v afegeix una línia de capçalera, amb el nom del fitxer

`$ head -n 2 -v quixot.txt`

==>quijote.txt <==

En un lloc de la Manxa, del nom del qual

no vull recordar-me, no hi ha molt de temps

tail

Mostra el final del fitxer.

Algunes opcions són:

- N N o -N mostra les últimes N línies (per defecte, 10)*
- +N mostra de la línia N al final*
- c N mostra els últims N bytes*
- f fa que tail corri en un llaç, afegint línies a mesura que el fitxer creix (útil per quan volem veure com es modifica un fitxer)*
- retry útil amb -f; encara que el fitxer no existeixi o sigui inaccessible continua intentant fins que el pot obrir*
- v afegeix una línia de capçalera, amb el nom del fitxer*

```
$tail -n 2 -v quixot.txt  
==>quijote.txt <==  
drassana, adarga antiga, rocín flac i  
llebrer corredor
```

grep

Es tracta d'una eina de cerca en fitxer mitjançant l'ús d'expressions regulars. Funciona de la següent manera:

\$ grep [-opcions] patró fitxers

Les opcions més comunes són:

- E o egrep: utilitza expressions regulars esteses*
- F o fgrep: interpreta els patrons no com a expressions regulars sinó com a cadenes de caràcters fixes*
- R o rgrep: llegeix tots els fitxers sota cada directori, recursivament*
- io --ignore-case: cerca ignorant diferències entre majúscules i minúscules*
- wo --word-regexp: per forçar que la cadena reconeguda sigui una paraula completa*

UD5. ADMINISTRACIÓ DE GNU/LINUX

- ho --fills-with-matxes: no mostra el contingut de la línia trobada però sí que mostra el fitxer que conté la cadena cercada*
- n anteposa a cada línia aparellada amb el seu número de línia (útil per buscar alguna cosa en un fitxer molt llarg i saber on aquesta específicament)*
- c imprimeix només el número de coincidència trobada*
- v només es busquen les no coincidències (quan el que busquem és el que no coincideix amb el patró)*

UD5. ADMINISTRACIÓ DE GNU/LINUX

El patró és qualsevol conjunt de caràcters que es cerca. Cal indicar que si van acompanyats d'un espai *grep* confondrà el patró amb els fitxers a cercar, això se soluciona amb l'ajuda dels caràcters". Per exemple:

```
$ grep "Hola món" fitxer
```

Si busquem alguna cadena que contingui un comodí, apòstrofs, cometes, redireccions o barres del tipus "\", caldrà anteposar una barra "\" per així indicar que busquem aquest caràcter en si i no la substitució del comodí, o que iniciem una cadena de diverses paraules. Per exemple:

```
$ grep \"*\" \"'\"?\"< fitxer
```

Això és una cadena xunga -> *"<

Per exemple, la següent ordre cercarà al fitxer tfons imprimirà totes les línies que no contenen números. D'aquesta manera sé si a la meva agenda de telèfons hi ha noms sense els telèfons corresponents.

```
$ grep -v ' [ 0 -9] ' tfons
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Per exemple, la següent ordre buscaria “Unix”, però no “UNIX” ni “unix”:

```
$ grep Unix fitxer
```

Posant:

```
$ grep -i Unix fitxer
```

cercaria totes les combinacions de majúscules i minúscules de “Unix”.

Per exemple:

```
$ grep -l pepe *
```

Tornaria el nom dels fitxers del directori actual que contenen el nom “*pepe*”.

UD5. ADMINISTRACIÓ DE GNU/LINUX

Altres exemples:

```
$grep cn
```

Cerca totes les ocurrencies que tingui una cadena amb una c ,
qualsevol lletra i una t

```
$ grep [Bc]el
```

Busca les ocurrencies Bel i cel.

```
$ grep [mo]ata
```

Dóna les línies que continguin mata, nata i oata.

```
$ grep [^mo]ata
```

Dóna les línies que contingui una cadena acabada en lliga i
que no contingui com a primera lletra m,m jo

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
$ grep "^Martin menja"
```

Com a sortida les línies que comencin amb Martin menja , ull vegeu que no està entre claudàtors , en aquest cas no és una negació de conjunt com a l'anterior exemple sinó com a començament de línia.

```
$ grep "dormint$"
```

Trobareu les línies acabades en dormint. \$ substitueix un fi de línia.

```
$ grep "^Caixa San Fernando guanya la lliga$"
```

Cerca aquelles línies que siguin exactament això.

Per descomptat aquests caràcters per eliminar el seu significat de substitució cal anteposar-los l'obligatòria "\". Per exemple:

```
$ grep "E\.T\."
```

Buscarà la seqüència ET

8. Scripts

Introducció

- El *shell* és un intèrpret d'ordres que permet a l'administrador executar determinades tasques.
- Però el *shell* no és únicament això, ja que els intèrprets de ordres són un autèntic llenguatge de programació que li permet al administrador automatitzar i programar tasques.
- Com qualsevol llenguatge de programació, el *shell* de GNU/Linux, incorpora sentències de control de flux, sentències de assignació, funcions, etc.
- Els programes de *shell* no necessiten ser complicats, com passa en altres llenguatges, i són executats línia a línia per el que a aquests programes se'ls coneix amb el nom de *shell*.

Introducció

- Des de que a els anys 70 es desenvolupara UNIX, es han inclòs amb ell diverses variants del llenguatge de shell.
- El més popular i comú és el *Bourne Shell*, per la seva creador.
- A les variants de UNIX de BSD es va incloure el *C-Shell*, una variant amb sintaxi més semblant a C que el Bourne.
- Tambien, el Korn *shell* incluíó funcionés para controlar els treballs a segon plànol, etc.
- Al cas de Linux, s'inclou el Bash (Bourne-again shell), que aglutina característiques de totes les variants, però que segueix la filosofia del Bourne.
- S'utilitzarà aquest intèrpret per ser el que ve per defecte.

Conceptes bàsics

Executar el nostre primer script

1. Crear el arxiu “primer_script.sh”

```
$ nano primer_script.sh
```

- El script deu incloure un “bang line”. Es tracta de la primera línia de un script, que ha de començar amb els caràcters `#!` Que especifica la ruta on es troba l'interpret.

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

- Mostrarem un missatge per pantalla amb el comandament “echo”
`echo “Hola Món! El meu primer script”`

Conceptes bàsics

Executar el nostre primer script

2. El arxiu ha de tindre permisos d'execució.

Per atorgar permisos de execució a el nostre script, hem de escriure:

```
$ chmod 755 primer_script.sh
```

3. Executar el script. Tenim 3 opcions:

- *\$ sh primer_script.sh*
- *\$ bash primer_script.sh*
- *\$./primer_script.sh*

Conceptes bàsics

Executar el nostre primer script

3. També podem optar per copiar el script a una ruta “especial” que permeti invocar-ho fàcilment. Per exemple, podem copiar-ho a */usr/sbin* i executar-ho des de qualsevol part sense incloure la ruta completa on es trobi:

- *\$ sudo cp primer_script.sh /usr/sbin/*

Executem *el script*:

- *\$ primer_script.sh*

Conceptes bàsics

Variables

- Com a qualsevol llenguatge de programació, les variables es utilitzen per poder guardar informació i a partir de ella poder prendre decisions o realitzar operacions.
- Lògicament, les variables no poden tenir el nom de cap paraules reservada (pe fet) i hi ha dos formes diferents de utilitzar-la depenent de si volem assignar-lo un valor u operar amb elles.
- A continuació anem a veure un exemple en què s'assigna a la variable nombre un valor i després es mostra per pantalla.

Conceptes bàsics

Variables

exemple1.sh

```
#!/bin/bash
numero=5
echo "el valor de la variable es "$numero
```

- Com es pot veure a el exemple quan es vol accedir al valor de la variable es utilitza el símbol \$.

UD5. ADMINISTRACIÓ DE GNU/LINUX

- Les variables les establim amb el símbol “=” sense espais en blanc.
- Per llegir-ne el contingut, ja sigui per mostrar-lo per pantalla o per comparar-lo amb algun valor hem de fer ús del símbol “\$”.

```
#!/bin/bash

miNumero=4
miString="Hola"
#si queremos guardar lo que nos devuelve un comando usamos comillas invertidas `
miResultadoComando=`pwd`

echo "Para ver el contenido de nuestras variables usamos el símbolo de $"
echo La variable $miResultadoComando contine lo devuelto por el comendo pwd
echo $miString tengo $miNumero años
```

```
miguel@miguel-VirtualBox:~$ ./ejemploVariables.sh
Para ver el contenido de nuestras variables usamos el símbolo de $
La variable /home/miguel contine lo devuelto por el comendo pwd
Hola tengo 4 años
```

- Si volem mostrar una variable i ajuntar-la amb un literal hem de fer ús de \${variable}.

Variables d'entorn:

Les variables d'entorn són variables del sistema. Podem accedir-hi mitjançant el comandament “env” encara que és recomanable fer-ho juntament amb l'ordre “grep” o “more” per poder fer una lectura concreta.

```
miguel@miguel-VirtualBox:~$ set | grep HOME
HOME=/home/miguel
local -a dirs=({BASH_COMPLETION_USER_DIR:-${XDG_DATA_HOME:-$HOME/.local/share}/bash-completion}/completions);
i="$HOME/.ssh/$i";
```

Variables d'entorn a destacar:

\$HOME -> Guarda la ruta del nostre directori home.

\$PATH -> Indica les rutes on cerca les ordres o fitxers d'aplicacions.

```
miguel@miguel-VirtualBox:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

\$IFS -> Separador de camps. Per defecte, és l'espai en blanc. Ho veurem en gran detall en exercicis amb bucles per a lectura de fitxers on el nostre separador per defecte passarà a ser el salt de línia.

\$UID -> UID de l'usuari actual.

\$GID -> GID de l'usuari actual.

\$HOSTNAME -> Nom de la màquina actual.

Crear una variable d'entorn:

1. Crear la variable.
2. Exportar la variable. Això serà vàlid fins al següent inici de sessió

```
miguel@miguel-VirtualBox:~$ MIGUEL="Mi variable de entorno"
miguel@miguel-VirtualBox:~$ env | grep MIGUEL
miguel@miguel-VirtualBox:~$ export MIGUEL
miguel@miguel-VirtualBox:~$ env | grep MIGUEL
MIGUEL=Miguel variable de entorno
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Modificar una variable d'entorn fins a final de sessió:

1. Modificar la variable: `PATH=$PATH:/home/sbin`
2. Exportar la variable: `export $PATH`.
 - a. Exportar ens permet que els canvis s'apliquin fins al final de la sessió.

```
miguel@miguel-VirtualBox:~$ PATH=$PATH:/home/sbin
miguel@miguel-VirtualBox:~$ export PATH
```

Canviar/Crear una variable d'entorn per arrencar un usuari.

1. Hem d'afegir la nostra entrada al fitxer `/home/"usuari"/.profile`

Modificar/Crear una variable d'entorn per arrencar tots els usuaris.

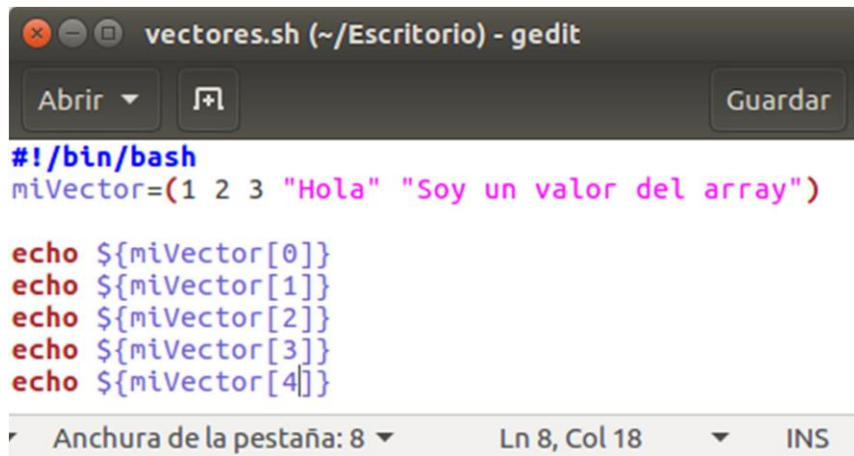
1. Hem d'afegir la nostra entrada al fitxer `/etc/environment`

```
miguel@miguel-VirtualBox:~$ cat /etc/environment
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

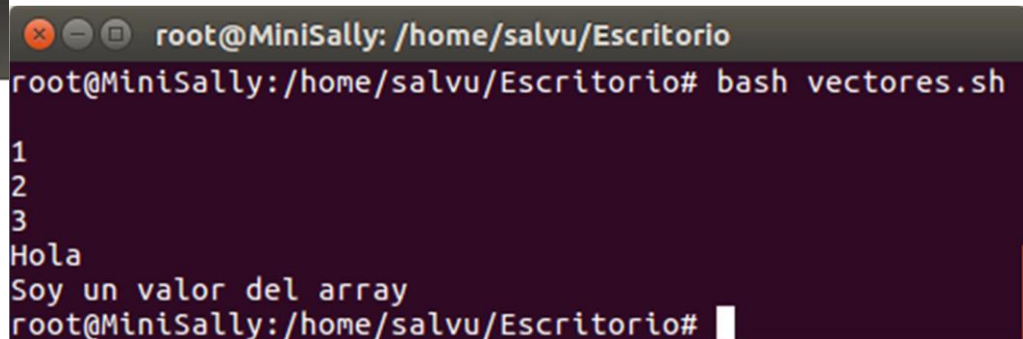
Vectors:

- Un vector es pot instanciar de manera manual.
`miVector=(valor1 valor2 valor3 ... valorN)`
- Es pot llegir el contingut de cada posició del vector de la manera següent:
`${miVector[posició]}`
- La posició del primer valor és 0 i el darrer és (N-1) on N és el nombre total d'elements.
- Exemple:



```
#!/bin/bash
miVector=(1 2 3 "Hola" "Soy un valor del array")

echo ${miVector[0]}
echo ${miVector[1]}
echo ${miVector[2]}
echo ${miVector[3]}
echo ${miVector[4]}
```

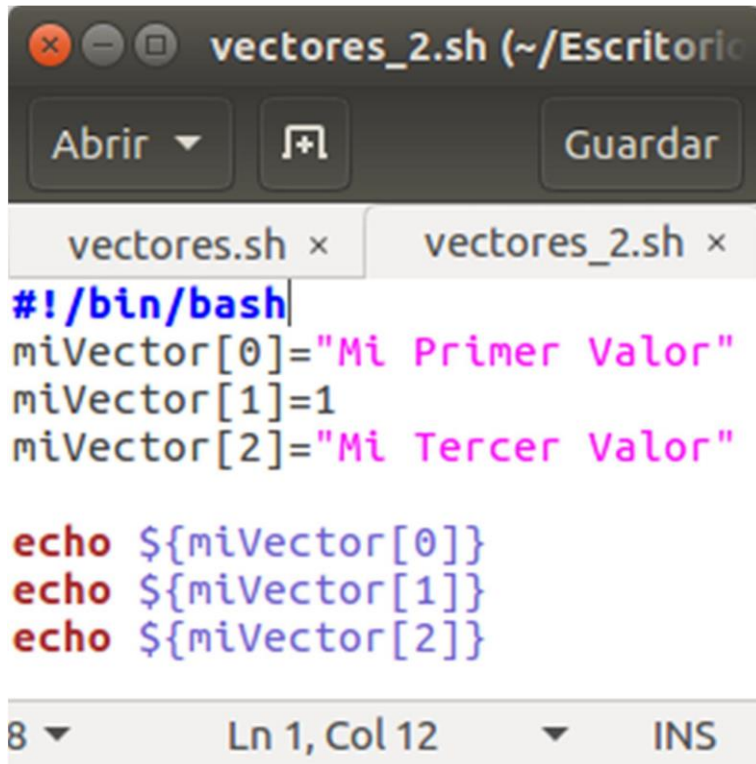


```
root@MiniSally: /home/salvu/Escritorio# bash vectores.sh
1
2
3
Hola
Soy un valor del array
root@MiniSally: /home/salvu/Escritorio#
```


UD5. ADMINISTRACIÓ DE GNU/LINUX

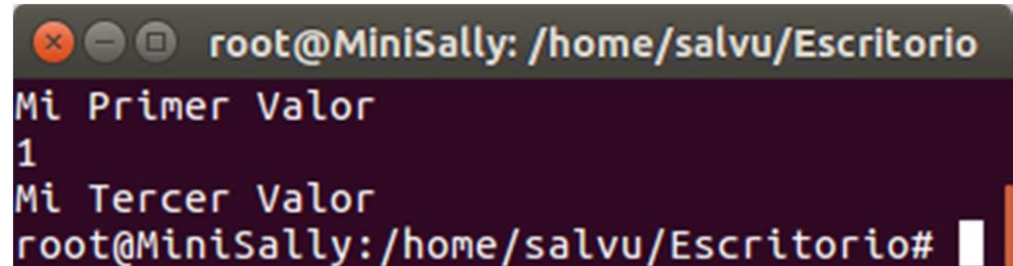
Podem sobre escriure o donar un nou valor a un dels components del vector:

`miVector[posició]=valor`



```
#!/bin/bash
miVector[0]="Mi Primer Valor"
miVector[1]=1
miVector[2]="Mi Tercer Valor"

echo ${miVector[0]}
echo ${miVector[1]}
echo ${miVector[2]}
```



```
root@MiniSally: /home/salvu/Escritorio
Mi Primer Valor
1
Mi Tercer Valor
root@MiniSally: /home/salvu/Escritorio#
```

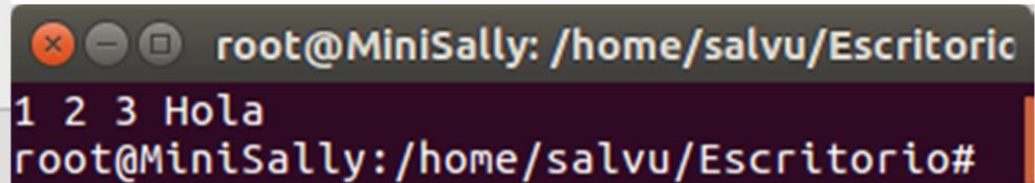
UD5. ADMINISTRACIÓ DE GNU/LINUX

Mostra tots els valors continguts en un vector:

```
${miVector[*]}
```



```
#!/bin/bash
miVector=(1 2 3 "Hola")
echo "${miVector[*]}"
```

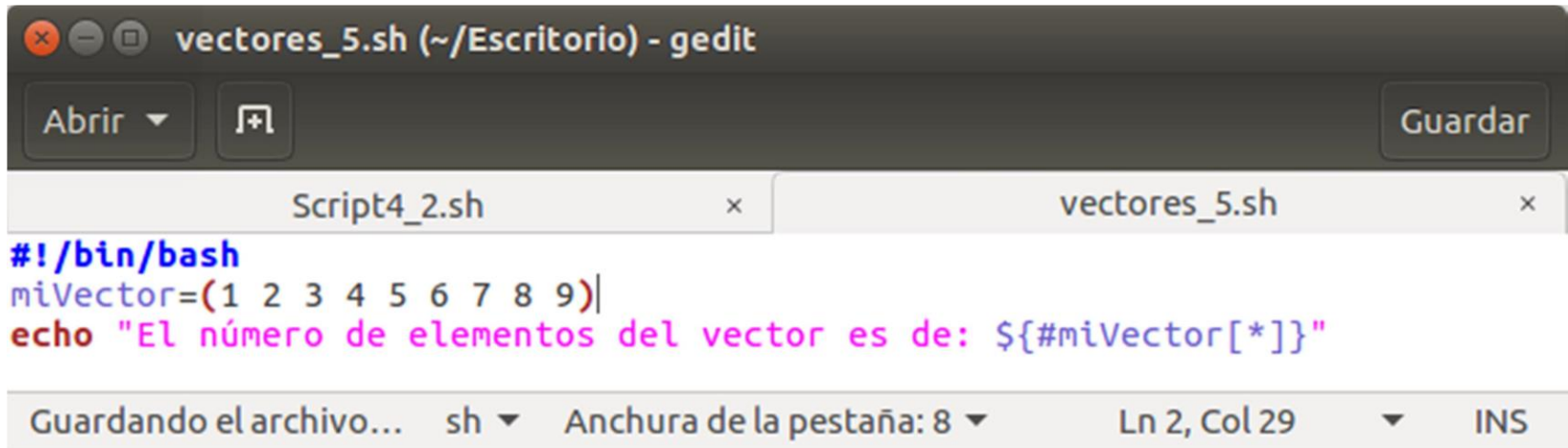


```
root@MiniSally: /home/salvu/Escritorio#
1 2 3 Hola
root@MiniSally: /home/salvu/Escritorio#
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Mostra el nombre d'elements d'un vector (longitud d'un vector):

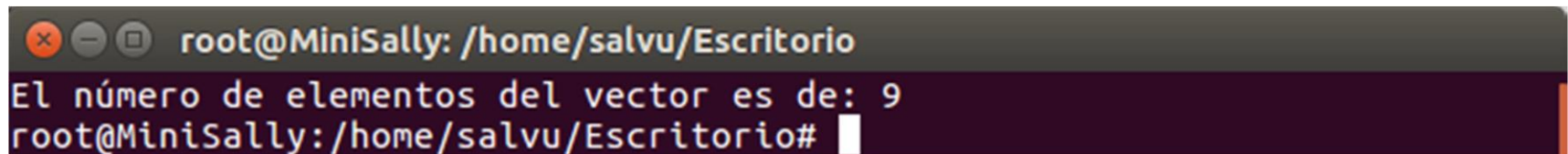
`${#el meuVector[*]}`



A screenshot of a gedit editor window titled "vectores_5.sh (~/Escritorio) - gedit". The window has a menu bar with "Abrir" and "Guardar" buttons. Below the menu bar, there are two tabs: "Script4_2.sh" and "vectores_5.sh". The "vectores_5.sh" tab is active, showing the following script content:

```
#!/bin/bash
miVector=(1 2 3 4 5 6 7 8 9)
echo "El número de elementos del vector es de: ${#miVector[*]}"
```

At the bottom of the editor, there is a status bar showing "Guardando el archivo...", "sh", "Anchura de la pestaña: 8", "Ln 2, Col 29", and "INS".



A screenshot of a terminal window with the prompt "root@MiniSally: /home/salvu/Escritorio". The terminal shows the output of the script:

```
El número de elementos del vector es de: 9
root@MiniSally: /home/salvu/Escritorio#
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Recórrer un array:

```
#!/bin/bash
miVector=(1 2 3 4 5 6)
for i in ${miVector[*]}
do
    echo $i
done
```

Arguments.

Hem de tindre clar que són: \$*, \$#, \$0 i \${1-9} i shift.

Els arguments van separats per espais

En cas d'introduir un argument que tingui espais, l'haurem d'entrecometes.

En cas de passar com a argument un, com *, cal escapar-ho “\”:

```
./soyUnScript.sh \*
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Exemple:

```
#!/bin/bash
echo -e "\$* es una lista con todos los argumentos pasados al script: $* \n"
echo -e "\$# es el número de argumentos: $# \n"
echo -e "\$3 es la forma de acceder al contenido del tercer argumento: $3 \n"
echo -e "\$0 es el menos usado, hace referencia al nombre del script: $0 \n"
```

```
ubuntu@salva:~$ ./ejemploArgumentos.sh 1 2 "Hola" "Hola y Adios" 6
$* es una lista con todos los argumentos pasados al script: 1 2 Hola Hola y Adios 6
$# es el número de argumentos: 5
$3 es la forma de acceder al contenido del tercer argumento: Hola
$0 es el menos usado, hace referencia al nombre del script: ./ejemploArgumentos.sh
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Exemple de shift:

```
echo -e "El comando shift hace que se elimine el primer argumento y el resto se muevan una "
echo -n "posición, es decir, el segundo argumento pasará a ser el primero"

echo -e "El valor de \$1 es:  \$1"
echo -e "El valor de \$2 es:  \$2"

shift

echo "Una vez hecho el shift vemos que se han movido."

echo -e "El valor de \$1 ahora es:  \$1"
echo -e "El valor de  \$2 ya no existe:  \$2"

El comando shift hace que se elimine el primer argumento y el resto se muevan una
posición, es decir, el segundo argumento pasará a ser el primero

El valor de $1 es:  A
El valor de $2 es:  B

Una vez hecho el shift vemos que se han movido.

El valor de $1 ahora es:  B
El valor de  $2 ya no existe:
ubuntu@calua:~$
```

Estructures de control

Condicció simple (IF)

- Les condicions simples (if) ens permeten que en cas de complir-se una determinada condició es executeu un determinat codi. La sintaxi de les sentències *if* és:

```
If [ condiciona ]  
  then  
    comandus  
  else  
    ordres  
fi
```


UD5. ADMINISTRACIÓ DE GNU/LINUX

A continuació en podem veure un exemple:

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

```
echo -n "Introduce un valor: "  
read var
```

```
if (( var < 10 ))  
then  
    echo "Es menor que 10"  
else  
    echo "Es mayor que 10"  
fi
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Important tindre en compte que cal deixar espais en blanc entre els claudàtors.

Totes les estructures de programació tenen una obertura i un tancament. En comptes de ser claus {} com estem acostumats són paraules reservades. En el cas de if `fi`.

Com podem veure es pot niar amb altres condicionals “if” mitjançant la estructura elif.

```
#!/bin/bash
if [ $# -gt 0 ]
then
    echo "Has pasado más de un argumento"
fi
```

```
#!/bin/bash
if [ $# -gt 0 ]
then
    echo "Has pasado más de un argumento"
else
    echo "No has pasado ningún argumento"
fi
```

```
#!/bin/bash
if [ $# -gt 1 ]
then
    echo "Has pasado más de un argumento"
elif [ $# -eq 1 ]
then
    echo "Has pasado 1 argumento"
else
    echo "No has pasado ningún argumento"
fi
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Condicions múltiples (CASE)

- Quan volem realitzar moltes condicions sobre un mateix valor (pe a un menú) la millor opció és utilitzar *case*.
- La seva sintaxi és:

```
case $variable in
    valor1)    comando
               --
               comando;;
    valor2)    comando
               --
               comando;;
```

...

```
*)          comando
            --
            comando;;
esac
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

A continuació en podem veure un exemple:

```
#!/bin/bash  
echo -n "Introduce un valor: "  
read var1
```

```
case $var1 in  
  1) echo " uno ";;  
  2) echo " dos ";;  
  3) echo " tres ";;  
  4) echo " cuatro ";;  
  *) echo "opcion incorrecta ";;  
esac
```

```
#!/bin/bash
read -p "Elige un número entre el 1 y el 3 o la letra y" num
case $num in
    0)
        echo "Has elegido el 0"
        echo "La última línea acaba en ;;";;
    1)
        echo "Has elegido el 1";;
    2)
        echo "El 2";;
    3)
        echo "El 3";;
    "y")
        echo "Letra y seleccionada";;
    *)
        echo "Cualquier otra cosa"
        echo "Recuerda que el ;; solo se pone en la última línea";;
esac
```

- El cas del case és molt usat per a la creació de menús interactius
- Es busca una coincidència amb el contingut de la variable “num”, si no fem servir cometes s'espera un valor numèric, si volem comparar-ho amb strings hem de fer ús de cometes.
- El valor “*” serveix com a valor per defecte en cas de no haver trobat coincidència.
- És important destacar que l'última ordre executada de cada opció ha de finalitzar amb doble punt i coma “;;”

UD5. ADMINISTRACIÓ DE GNU/LINUX

Es poden niar diverses opcions en una, com si féssim un “or”. Exemple:

```
#!/bin/bash

read -p "Elige un nombre: " nombre

case $nombre in
    alvaro | pepe)
        echo "Tu nombre es alvaro o pepe";;
    *)
        echo "Otro nombre";;
esac
```

```
miguel@miguel-VirtualBox:~$ ./ejemploCase.sh
Elige un nombre: alvaro
Tu nombre es alvaro o pepe
miguel@miguel-VirtualBox:~$ ./ejemploCase.sh
Elige un nombre: miguel
Otro nombre
```


Bucle for

- El bucle *for* s'utilitza per a executar un codi un determinat número de vegades.
- La seva sintaxi és:

```
for variable in llista_elements do  
    accions  
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

A continuació en podem veure un exemple:

```
#!/bin/bash

for i in `seq 5 15`
do
    echo $i
done
```

- També es pot utilitzar el *for* per moure's a una llista de elements:

```
for variabe in llista_elements
do
    accions
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

- A continuació podem veure un exemple (llistar la informació DNS de dos dominis introduïts per teclat).

```
#!/bin/bash

echo -n "Introduce el primer dominio: "
read dom1

echo -n "Introduce el segundo dominio: "
read dom2

for HOST in $dom1 $dom2
do
    echo $HOST
    /usr/bin/host $HOST
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
#!/bin/bash
for miVariable in 1 2 3 4 5 6
do
    echo $miVariable
done
```

```
#!/bin/bash
for miVariable in $*
do
    echo $miVariable
done
```

```
#!/bin/bash
listado="a b c d e g"
for miVariable in $listado_
do
    echo $miVariable
done
```

Bucle while

- El bucle *while* permet executar un codi fins que no es compleixi una determinada condició de sortida
- La seva sintaxi és:

```
while [ condició ]  
do  
    accions  
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

- A continuació podem veure un exemple (llegir dades de teclat fins que es premi un "0").

```
#!/bin/bash

var=1

echo -n "Introduce un número (0 para salir)"
read var

while [ $var != 0 ]
do
    echo $var
    echo -n "Introduce un número (0 para salir)"
    read var
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
#!/bin/bash
while [ true ]
do
    echo "BUCLE INFINITO"
done
```

```
#!/bin/bash
read -p "Introduce un número " num
while [ $num -ne 0 ]
do
    echo "El número elegido no es 0"
    read -p "Introduce un número " num
done
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Until:

```
#!/bin/bash
a=0
until [ $a -eq 10 ]
do
    a=`expr $a + 1`
    echo $a
done
```


Funcions

- Una **funció** és un bloc de codi que permet la seva reutilització de una forma fàcil i senzilla.
- Es recomana que el nom de la funció sigui ho més descriptiu possible i que descriviu ho més fidelment possible el funcionament de la funció.
- Per definir una funció es fa de la següent forma:

nom_funcio()

- A continuació, es declara la instrucció o conjunt instruccions que s'executaran cada vegada que es realitza la crida a la funció, com sempre entri claus.

12.5 Funcions

- exemple.sh: realitzar la suma de dos valors introduïts per teclat.

```
#!/bin/bash

suma()
{
    resultat= `expr $a + $b`
    echo "a + b =" $resultado
}

echo -n "Introduce el valor de a: "
read a
echo -n "Introduce el valor de b: "
read b

suma $a $b
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

```
#!/bin/bash

DevuelveCinco(){
    echo "Hola voy a devolver un 5"
    return 5
}

#Esto ya es parte del programa principal que invoca a la función

DevuelveCinco

#El valor de $? es el valor devuelto por el último comando ejecutado.
#En nuestro caso ha sido nuestra función.

echo "El valor devuelto es 5"
```

```
miguel@miguel-VirtualBox:~$ ./ejemploFuncion.sh
Hola voy a devolver un 5
El valor devuelto es 5
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

- Les funcions poden tornar resultats numèrics entre 0 i 255.
- El valor retornat es desa a la variable “\$?”. Aquesta variable és sobreescrita cada vegada que executem una ordre o funció.

```
#!/bin/bash

DevuelveCinco(){
    echo "Hola voy a devolver un 5"
    return 5
}

#Esto ya es parte del programa principal que invoca a la función

DevuelveCinco

#El valor de $? es el valor devuelto por el último comando ejecutado.
#En nuestro caso ha sido nuestra función.

echo "El valor devuelto es 5"

#ejecutamos un nuevo comando
pwd
#Comprobamos que $? tiene un nuevo valor
#Habitualmente el valor 0 indica que el comando se ha ejecutado correctamente

echo $?
```

```
miguel@miguel-VirtualBox:~$ ./ejemploFuncion2.sh  
Hola voy a devolver un 5  
El valor devuelto es 5  
/home/miguel  
0
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

Per això és interessant que després d'executar la nostra funció ens guardem el valor de “\$?” en una variable:

```
#!/bin/bash
DevuelveCinco(){
    echo -e "Hola voy a devolver un 5"
    return 5
}

#Esto ya es parte del programa principal que invoca a la función.
DevuelveCinco

#Nos guardamos $? en una variable
miValorRetornado=$?

#Ejecutamos un nuevo comando
pwd

#Comprobamos que $? tiene un nuevo valor.
#Habitualmente el valor 0 indica que el comando se ha ejecutado correctamente.
echo -e "El valor de \ $? es $? , pero el de la función es $miValorRetornado"
```

```
Hola voy a devolver un 5
/home/ubuntu
El valor de $? es 0 , pero el de la función es 5
```

UD5. ADMINISTRACIÓ DE GNU/LINUX

- Si volem passar paràmetres no cal declarar-los a la funció, simplement en fem ús i els recollim igual que els arguments d'un script: (\$*, \$#, \$1, \$2, \$3...)

```
#!/bin/bash

DevuelveCinco(){
    echo "Nos han pasado $# parámetros"
    echo "Los parámetros pasados son $*"
    echo "El segundo parámetro es $3"
}

#Esto ya es parte del programa principal que invoca a la función
DevuelveCinco "Hola" 1 2 3
```

```
miguel@miguel-VirtualBox:~$ ./ejemploFuncion3.sh
Nos han pasado 4 parámetros
Los parámetros pasados son Hola 1 2 3
El segundo parámetro es 2
```