



ÍNDEX

1. COMANDES AL SISTEMA OPERATIU LINUX.....	5
SHELL.....	5
Històric d'ordres.....	6
Les variables del shell.....	7
Variables predefinides.....	7
La Variable PATH.....	8
Àrees de dades.....	9
Ordres relacionades amb l'entorn.....	10
Metacaràcters.....	10
Substitució d'ordres i alies.....	12
Redireccions.....	12
Personalització del Prompt.....	13
Ordres bàsiques.....	15
exit:.....	15
who [am i]:.....	16
write:.....	16
mesg:.....	16
date:.....	16
cal:.....	16
uname:.....	17
passwd:.....	17
clear:.....	18
script:.....	18
man:.....	18
Ordres del sistema d'arxius.....	18
cd:.....	19
ls:.....	19
pwd:.....	21
mkdir:.....	21
rmdir:.....	21
Editor de text: nano.....	21
Ús d'arxius i permisos.....	22
cat:.....	22
head:.....	22
tail:.....	22
cp:.....	22
mv:.....	23
rm:.....	23
file:.....	23
touch:.....	23
dd:.....	24
chmod:.....	24
umask:.....	25
whereis:.....	25
id:.....	25
su:.....	26



sudo:	26
Expressions regulars	26
Filtres	27
grep:	27
sort:	29
tr:	31
cut:	32
Pipelines	33
Cercar fitxers	34
find:	34
locate:	36

1. COMANDES AL SISTEMA OPERATIU LINUX

SHELL

Per a fer servir eines textuais a Linux fem servir l'interpret d'ordres. L'interpret d'ordres és una interfície entre l'usuari i el sistema operatiu basat en ordres escrites i teclejades en aquest interpret. Ja l'hem fet servir en pràctiques anteriors i ara intentarem conèixer més profundament les seves característiques. Ens referirem a ell també com a *shell*.

A Linux hi ha diversos tipus d'interprets d'ordres o *shells*, cadascun amb les seves particularitats, si bé la semblança és molt gran i compleixen tots amb una sèrie de funcions comunes. Aquestes funcions són:

- Verifica si una ordre es interna del *shell* o és externa (un programa executable, cas en que haurà de buscar-la amb l'ajuda de la variable PATH).
- Realitza la substitució d'ordres (àlies).
- Gestiona la redirecció de sortida/entrada, d'error i les canonades (*pipelines*) (`>`, `>>`, `<`, `2>`, `2>>`, `|`).
- Genera noms d'arxiu a partir de metacaràcters (`*`, `?`, etc)
- Substitueix variables d'entorn pel seu valor.

El *shell* té les següents formes d'invocar una ordre:

Ordre &	Executa l'ordre en segon pla. Mentre s'executa el cursor torna a estar disponible per teclejar més ordres. Observeu la diferència entre <code>gedit</code> i <code>gedit &</code>
Ordre1 ; Ordre2	Pots executar moltes ordres en una sola línia. S'executen una després de l'altra. Per exemple: <code>echo hola ; sleep 5 ; echo mon</code>
Ordre1 Ordre2	La sortida de l'ordre 1 es redirigeix cap a l'ordre 2, enlloc d'anar cap a la pantalla. Per exemple: <code>ls -l wc -l</code> . Aquesta comanda llista un directori i ho reenvia cap al comptador de paraules. El resultat és que ens diu quants fitxer i carpetes hi ha al directori llistat. A l'operador <code> </code> se l'anomena <i>pipeline</i> .
Ordre1 `Ordre2`	La sortida de l'Ordre2 s'utilitza com argument de l'Ordre1. Per exemple: <code>cat `ls`</code> - que ens mostrarà el contingut de tots els fitxers del directori actual <code>file `ls`</code> - mostrarà els tipus de fitxers de tots els fitxers del directori actual <code>wc -l `ls`</code> - que és el mateix que <code>ls -l wc -l</code> .
Ordre1 && Ordre2	Executa Ordre1 i si resulta amb èxit (no produeix cap error) executa l'Ordre2. Per exemple: <code>rm fitxer && ls</code> Aquesta comanda esborra <code>fitxer</code> , si no pugues <code>ls</code> no s'executaria; si l'esborres correctament executaria <code>ls</code> . <code>cp && echo hola</code> La segona part d'aquesta comanda no s'executaria perquè la comanda <code>cp</code> necessita paràmetre.
Ordre1 Ordre2	Executa Ordre2 només si Ordre1 falla. Per exemple: <code>rm fitxer </code>

`echo error` Aquesta comanda, si `fitxer` no existeix o no el pot esborrar, executarà `echo error`.

Històric d'ordres

Totes les ordres que anem invocant des de l'interpret són emmagatzemades amb l'objectiu de poder després repetir-les o modificar-les. Per a veure un històric d'ordres executarem l'ordre:

```
armand@1DAW:~$ history | head
 1 nano /etc/hostname
 2 exit
 3 cd
 4 ls
 5 clear
 6 ifconfig
 7 ping 192.168.0.1
 8 history
 9 wget informatica.iesjoaquimmir.cat/practiques.html
10 ooffice
armand@1DAW:~$ !6
ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:36:39:16
            inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::a00:27ff:fe36:3916/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:1111 errors:0 dropped:0 overruns:0 frame:0
            TX packets:637 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:868867 (868.8 KB)  TX bytes:66921 (66.9 KB)

lo          Link encap:Local Loopback
```

Per a repetir una ordre anterior sempre podem utilitzar els cursors amunt/avall. Però si l'ordre és molt anterior, amb l'ajut de l'històric i teclejant `!` seguit del número de posició de l'ordre, tal com es mostra a la imatge.

També en lloc del número podem posar una part de l'ordre, per exemple, si el resultat de la comanda `history` fos la següent:

```
1 cd
2 pwd
3 cp /etc/passwd .
4 ll
5 md copia
6 cp passwd copia
7 mv copia paraula
8 ll
9 ls /etc
10 ls /
11 ll /
12 ls /usr/bin
```

Podríem fer servir les següents combinacions:

`$!mv` – executarà l'ordre 7.

`$!ls` – executarà la darrera que comenci per `ls`, l'ordre 12.

`$!cp /e` – executarà `cp passwd copia /e` (Observa: No executa l'ordre 3, sinó la 6 a la qual li afegeix la cadena `/e`).



\$!?cp /e? - executarà la primera ordre que comenci per cp /e (número 3). Observa que els ? delimiten la cadena. No és obligatori posar-lo al final si no hi has de posar res més. Per tant podríem posar !? cp /e

Les variables del shell

De la mateixa forma que en programació, el *shell* també pot tenir variables. No cal declarar-les, i per defecte tenen totes el valor NULL. La forma de crear i donar valor a una variable és teclejant simplement:

```
usuari@X:~> x=37
usuari@X:~> cadena=hola
```

Cal assegurar-se que entre l'igual i la variable i el seu nom no hi ha espais. Si féssim això:

```
usuari@X:~> x = 37
bash: x: command not found
```

Per després utilitzar les variables del *shell* cal afegir el signe \$ abans del nom de la variable:

```
usuari@X:~> echo $x
```

Per esborrar una variable simplement:

```
usuari@X:~> x=
```

Podem utilitzar també el comandament `unset` per esborrar les variables.

```
usuari@X:~> unset x
```

`unset` té un altre us, que és el d'esborrar, si no se li passen paràmetres, totes les variables de l'àrea local. Més endavant comentarem que són les variables de l'àrea local.

Variables predefinides

Existeixen un grup de variable predefinides pel *shell* i que poden ser útils a l'hora de crear scripts.

Aquestes són:

- **HOME** → Defineix el directori personal, també anomenat de treball o d'arrancada.
- **PATH** → Determina en quins directoris ha de buscar les ordres i programes executables.
- **PS1** → Defineix el *prompt* del *shell* principal. El *prompt* que tenim per defecte ens permet veure el text `usuari@nomequip:ruta$`. Si fem, per exemple: `usuari@nomequip:ruta$ PS1="aaaa>"` Canviarà el *prompt* per: `aaaa>`
- **TMOUT** → Si té un valor més gran que zero, defineix el temps que es trigarà a tancar el *shell* si no hi ha activitat per part de l'usuari. Per exemple si val 5, als cinc segons la finestra del *shell* es tancarà si no hem introduït cap ordre. El fet d'escriure sense teclejar *return* no posa a 0 el comptador.
- **\$** → Conté el PID del procés que està executant el *shell* actual. Per veure'l em d'entrar `echo $$` (el primer dolar per fer referència a que llegim el valor de la variable, i el segon \$ és la variable en si). Hi ha moltes més variables predefinides.

La Variable PATH

El **PATH** (ruta) és una llista de directoris que mira el sistema operatiu a la recerca dels comandaments que introduïm a la consola. Normalment aquesta variable es defineix a l'inici del sistema, en un script, que pot

anar canviant amb el temps i les versions o distribucions. Per exemple, clàssicament aquesta variable es definia a `/etc/profile` i a `.bash_profile` o `.bashrc` (on encara es defineix moltes variables d'entorn) en el nostre directori d'inici. Altres versions més actuals ho defineixen al fitxer `/etc/login.defs`. Sigui com sigui, podem modificar-ho en qualsevol d'aquests.

Si una ordre no està a la ruta definida per `PATH`, haurem d'introduir una ruta completa quan escrivim la comanda o bé canviar al directori on es troba - cap de les dues resulten tan convenientes com introduir la comanda i confiar en `bash` per saber on buscar.

Un exemple: volem executar una fictícia ordre anomenada `pcllista`, que es troba al directori `/tmp/`. Si llistem el valor de la variable `PATH` ens mostrarà:

```
usuari@ubuntu:~$ echo $PATH
/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

El directori `/tmp/` no es troba definit en aquesta variable, per tant si executem:

```
usuari@ubuntu:~$ pcllista
pcllista: no s'ha trobat l'ordre
```

En canvi si executem

```
usuari@ubuntu:~$ /tmp/pcllista
resultat de la comanda
...
```

Ara l'ordre s'executarà ja que li hem donat la ruta completa.

Ho podem resoldre modificant la variable `PATH`. Si fem:

```
usuari@ubuntu:~$ PATH=/tmp
usuari@ubuntu:~$ pcllista
resultat de la comanda
...
```

Ara la variable `PATH` té el valor de `/tmp/`. Però aquí hi ha un problema. Fixeu-vos:

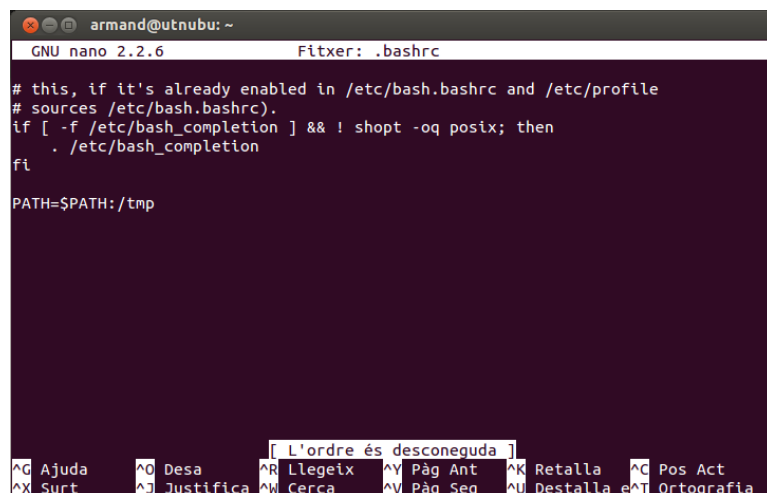
```
usuari@ubuntu:~$ cp
cp: no s'ha trobat l'ordre
```

L'ordre `cp` està disponible a `/bin/cp`. No s'ha trobat l'ordre perquè `/bin` no és a la variable d'entorn `PATH`. Hem esborrat les altres rutes de la variable `PATH` i ara no pot executar cap altra ordre, sense posar-hi la ruta completa. Tanquem el terminal i el reobrim. La variable `PATH` tornarà a agafar el seu valor inicial (i per tant `/tmp/` no hi serà). Ara ho farem d'una forma millor per a evitar que la variable `PATH` perdi la informació que ja té:

```
usuari@ubuntu:~$ PATH=$PATH:/tmp
usuari@ubuntu:~$ pcllista
resultat de la comanda
...
usuari@ubuntu:~$ cp
cp: manca un operand fitxer
Proveu «cp --help» per a obtenir més informació.
```

Hem afegir el valor `/tmp/` a la variable `PATH`, tenint en compte que cada ruta està separada de l'altra per «:».

Fixeu-vos que cada cop que obrim i tanquem el terminal aquest valor es perd. Si modifiquem el fitxer `.bashrc`, per exemple, i afegim al final el que veus a la imatge, la variable `PATH` sempre tindrà el valor actualitzat quan obris un terminal



Àrees de dades

Hi ha dues àrees de memòria incorporades al *shell* per desar les variables. Aquestes són: l'àrea local de dades i l'entorn . Per defecte una variable sempre es desa a l'àrea local. Aquestes variables són “privades”, associades al *shell* actual, és a dir, qualsevol subprocés fill no tindrà accés a aquesta variable. Per exemple:

```
usuari@X:~> NUMERO=34
usuari@X:~> echo $NUMERO
34
usuari@X:~> bash
usuari@X:~> echo $NUMERO

usuari@X:~> exit
exit
usuari@X:~> echo $NUMERO
34
usuari@X:~>
```

En obrir un altre *shell* (ordre `bash` o `gnome-terminal`, si no estem en un servidor), fill del *shell* principal (`gnome-terminal` no crea un procés fill, però còpia les seves variables d'entorn), la variable `NUMERO` no existeix.

Aquí hem obert un altre *shell*, però seria semblant a utilitzar un programa que l'executéssim des d'aquest *shell* i que necessites variables, les de l'àrea local no les veuria. Caldria exportar-les a l'entorn.

Ordres relacionades amb l'entorn

- `export` → Sense paràmetres mostra totes les variables de l'entorn. Quan li passem com a paràmetre un nom de variable, aquesta és exportada des de l'àrea local a l'entorn.

Exemple:

```
usuari@X:~> NUMERO=34
usuari@X:~> export NUMERO
```



```
usuari@X:~> echo $NUMERO
34
usuari@X:~> bash
usuari@X:~> echo $NUMERO
34
usuari@X:~> exit
exit
usuari@X:~> echo $NUMERO
34
usuari@X:~>
```

En aquest cas, el subprocés sí tindria accés a la variable.

- `unset` → Permet esborrar les variables i sense argument ens esborra totes les variables de l'àrea local.
- `set` → Llista totes les variables d'entorn i de l'àrea local.

Metacaràcters

Els metacaràcters o comodins ens permeten referenciar arxius o carpetes d'una forma més compacta sense necessitat d'escriure tots els noms complets. Si per exemple volem esborrar tots els arxius d'un directori que acaben en `.tmp`, en lloc d'escriure els noms un a un, podem utilitzar els metacaràcters per indicar que només els que acabin en `.tmp` s'esborrin utilitzant una única expressió més compacta.

?	<p>Substitueix a qualsevol caràcter, i només un, excepte el primer punt (és a dir, no val per substituir el primer punt dels arxius ocults).</p> <p>Exemple:</p> <pre>usuari@equip:~> ls -d ????</pre> <p>BNR2 dire prof</p> <p>Llista els directoris amb només quatre lletres. No llista els que tenen tres o menys caràcters al seu nom. Tampoc els directoris ocults amb quatre lletres.</p> <p>Un altre exemple</p> <pre>usuari@equip:~> echo ????</pre> <p>aaaa BNR2 dire prof</p> <p>En aquest exemple llista directoris i arxius de quatre lletres del directori actual (observa que hem utilitzat l'ordre <code>echo</code>, fes servir <code>ls</code> i comprova la diferència).</p> <p>Per a entendre millor això pensa que <code>????</code> es substitueix per tots els arxius/directoris (del directori on estiguis) de 4 lletres. Per tant, estaríem executant: <code>echo aaaa BNR2 dire prof</code>, que donaria com a resultat <code>aaaa BNR2 dire prof</code>.</p>
[]	<p>Defineix un rang de caràcters. Dins es pot utilitzar el guió per delimitar un rang de caràcters i una admiració per a negar el que vingui després.</p>

Exemples:

`ls [a-n]???` - Llistarà tots els arxius que comencin per qualsevol lletra de la “a” a la “n” i que tinguin un total de quatre lletres. El resultat pot ser un pel sorprenent si esperes que només es llistin els arxius que comencin per lletres *a* a *n* minúscules.

`ls [!a-d]???` - Llistarà tots els arxius que no comencin per lletres que van de *a* a *d* i que tinguin un total de quatre lletres al seu nom. També el resultat aquí pot ser sorprenent

`ls [!a-z]????` Llistarà tots els arxius que no comencin per les lletres que van de la *a* la *z* i que tinguin un total de 5 lletres, el resultat com l'anterior no és el que sembla.

En la següent figura observa què es llista:

```
armand@utnubu: ~/proves
armand@utnubu:~/proves$ touch a1 a2 A1 b1 B2 c3 C3 d4 D4
armand@utnubu:~/proves$ ls [a-d]*
a1 A1 a2 b1 B2 c3 C3 d4
armand@utnubu:~/proves$
```

Més exemples:

`ls [abcdef]*` - Llistar arxius que comencin per a, b, c, d, e o f minúscules

`ls [!abcdef]*` - Llistar arxius que NO comencin per a, b, c, d, e o f minúscules

Dintre dels corxets podem emprar les anomenades classes en un format `[:classe:]`. Les possibles classes són: `alnum`, `alpha`, `ascii`, `blank`, `cntrl`, `digit`, `graph`, `lower`, `print`, `punct`, `space`, `upper`, `word`, i `xdigit`.

El seu significat el pots buscar a l'ajuda de Linux.

Exemples:

`ls [[:lower:]]*` - llistarà tots els arxius que comencin per lletra minúscula.

`ls [[:lower:]]*[[:upper:]]` - llistarà tots els arxius que comencin per minúscula i acabin per majúscula.

*

Substitueix 0 o més caràcters, excepte un primer punt. Exemples:

<code>ls *c</code>	Llistarà tots els arxius que acabin en c, incloent l'arxiu anomenat “c”, si existeix.
<code>ls a*c</code>	Llistarà tots els arxius que comencin per “a” i que acabin per “c”.
<code>ls [a-p]*</code>	Tots els arxius que comencin per una lletra entre la “a” i la “p”.

Substitució d'ordres i alies

La substitució d'ordres ens permet captar la sortida d'una ordre i assignar-la a una variable, o bé, utilitzar aquesta sortida com un argument d'una altra ordre. La forma de fer-ho és utilitzant l'accent obert (`) tancant l'ordre.

Exemples:

```
usuari@X:~> fecha=`date`  
usuari@X:~> echo $fecha  
Mon May 10 17:31:33 CEST 2014  
usuari@X:~>
```

Els àlies s'utilitzen per poder invocar a les ordres amb un nom diferent a l'utilitzat habitualment:

```
alias dir = "ls -ld"  
alias dirocult = "ls -a"
```

Exemples:

Per llistar tots els alies, utilitza l'ordre `alias` sense arguments.

Per esborrar un alies, l'ordre `unalias` seguit del nom de l'àlies a esborrar. Per exemple:

```
unalias dirocult
```

esborrarà l'alies creat.

Redireccions

És una de les característiques més interessants i versàtils del sistema Linux. Quan iniciem l'interpret d'ordres s'obren 3 arxius:

- `stdin`: Arxiu estàndard d'entrada. És el lloc des d'on els programes llegeixen les entrades (en general és el teclat).
- `stdout`: Arxiu estàndard de sortida. És el lloc a on els programes envien els resultats (en general la pantalla).
- `stderr`: Arxiu estàndard d'error. És el lloc a on els programes envien les seves sortides d'error (en general la pantalla).

Podem modificar el comportament estàndard del sistema amb les redireccions i aconseguir que les sortides dels programes, en lloc de mostrar-se per pantalla, s'emmagatzemin a un arxiu o s'enviïn directament a una impressora o les entrades dels programes en lloc de prendre-les des del teclat s'agafin des d'un arxiu.

Per a fer-ho tenim les següents redireccions:

- Redirecció d'entrada: aquesta redirecció ens permet prendre l'entrada d'un altre dispositiu en lloc del teclat. El més habitual és fer servir arxius per a incorporar la nova entrada. El símbol de redirecció d'entrada es `<` Ex: `write jbosch < text.txt`

- Redirecció de sortida: Aquesta redirecció ens permet donar la sortida a un altre dispositiu en lloc de la pantalla. Es fa amb `>`. En general la sortida és un arxiu, encara que també pot ser una impressora. En el cas dels arxius.
 - Si no existeix es crea.
 - Si existeix perd el seu contingut. Si volem mantenir el contingut de l'arxiu i afegir al final d'aquest la sortida haurem de fer servir l'operador `>>`

```
date > prueba
who >> prueba
cat arxiu1 arxiu2 > arxiu3
```

- Redirecció d'errors: a Linux les comandes produeixen diagnòstics d'error. Podem redireccionar aquests errors fent servir `2>` (redirecciona l'error a un fitxer eliminant el seu contingut previ) o `2>>` (redirecciona l'error a un fitxer sense eliminar el seu contingut previ).

```
david@trasto:~$ cp
cp: falta un archivo como argumento
Pruebe 'cp --help' para más información.
david@trasto:~$ cp 2> basura
david@trasto:~$ cat basura
cp: falta un archivo como argumento
Pruebe 'cp --help' para más información.
david@trasto:~$
```

Si volem evitar els missatges d'error sense necessitat d'haver de guardar-los en un fitxer podem redireccionar al dispositiu `/dev/null`, a on tot desapareix. Per exemple:

```
david@trasto:~$ cp
cp: falta un archivo como argumento
Pruebe 'cp --help' para más información.
david@trasto:~$ cp 2> /dev/null
david@trasto:~$
```

Personalització del Prompt

El *prompt* és el text recurrent que apareix a l'esquerra d'on anem a començar a escriure la comanda. Tampoc és absolutament necessari en l'actualitat. Abans que els escriptors es convertissin en la norma, l'indicador contenia informació útil que necessitàvem saber d'un cop d'ull, com el directori actual, la data i l'hora. Avui en dia, que podem aconseguir tota aquesta informació de l'escriptori, no hauríem de preocupar-nos massa per ell.

En canvi, fins i tot avui dia, trobarem útil disposar d'informació bàsica visible sempre, com el compte actual i el directori, fins i tot malgrat que utilitzem gairebé sempre un terminal virtual. Mai sabem quan necessitarem reparar el nostre sistema i no tinguem disponible l'escriptori. A més, si mirem algunes distribucions, trobarem sovint que el caràcter final a l'indicador ens diu que teniu un usuari normal o un compte de *root*, els *prompts* del compte de l'usuari acaben en un signe dòlar, mentre que els de *root* ho fan en un signe coixinet (`#`).

De fet, un cop que el mirem, veurem que distribucions diferents tenen *prompts* diferents basant-se en quina informació pensen que voldran veure els usuaris. Per exemple, si estigués treballant a `/usr/share` en un ordinador anomenat *equip*, a Fedora, el meu indicador predeterminat serà `[usuari@equip share]$`, mentre que a Debian serà `usuari@equip:/usr/share$`. D'aquesta diferència hauríem de deduir que els valors predeterminats de Fedora suposen que els usuaris romanen gran part del temps en els seus directoris d'inici perquè no se'ls dona el camí complet, mentre que els de Debian suposen que els seus usuaris més avançats

poden estar en qualsevol part del sistema i prefereixen no haver de fer servir la comanda `pwd` per a descobrir on es troben.

A més, possiblement desitgem escurçar l'indicador, especialment si estem llistant rutes a directoris anidats a cinc o sis nivells de profunditat. També pot ser que vulguem canviar el color del *prompt* per fer-lo més visible, per complaure el nostre sentit de l'estètica, o per ressaltar millor la diferència entre el compte de root i totes els altres.

Sigui quina sigui la nostra raó, el millor lloc per començar si volem canviar el nostre indicador és canviar temporalment el paràmetre indicador `PS1`. Aquests canvis romandran efectius fins que els canviem de nou o tanquem la nostra consola virtual actual. El terminal següent que obrim ens tornarà a l'indicador predeterminat.

Per començar, desitjarem veure les configuracions actuals per `PS1`, la variable per l'indicador, introduint la comanda: `echo PS1`

Probablement obtindrem una resposta que diu alguna cosa com: `[\u@\h\W]$`, que és la lectura que obtenim amb una màquina amb Fedora 10 instal·lat. A Ubuntu, per exemple obtindrem un resultat com: `${debian_chroot:+($debian_chroot)}\u@\h:\w$`

Comparant amb l'indicador de Fedora esmentat abans, podem entendre el que significa cada entrada. Noteu, però, els elements extra, com ara la `@` entre el nom d'usuari (`\u`) i el nom de l'ordinador/host (`\h`) i l'espai entre el nom d'amfitrió i el directori actual (`\W`).

Podem canviar aquest indicador temporalment prenent com a referència la següent taula.

<code>\d</code>	Data
<code>\h</code>	Nom de la màquina
<code>\t</code>	Hora actual en format 24 hores HH:MM:SS
<code>\T</code>	Hora actual en format 12 hores HH:MM:SS
<code>\@</code>	Hora actual en format 12 hores am/pm
<code>\A</code>	Hora actual en format 24 hores HH:MM
<code>\u</code>	Nom d'usuari de l'usuari actual
<code>\w</code>	Directorio de treball actual, amb el directori de nivell superior indicat amb una tilde (~)
<code>\W</code>	Nom base del directori de treball actual, amb el directori de nivell superior indicat amb una tilde (~)
<code>\l</code>	Barra invertida

No estan totes les opcions, però fent una recerca en algun cercador no trobarem massa problemes per a completar aquesta taula.

Per exemple, si volguéssim presentar el compte d'usuari actual, fariem un canvi temporalment executant `PS1="\u$ "` per obtenir l'indicador usuari\$. Nota: l'espai després del signe dòlar no és un error, sinó que s'afegeix deliberadament per a que el que escric estigui separat del *prompt*.

Si volem afegir color, podem canviar el dels caràcters d'acord amb la fórmula: `"\e[x;ym\e[m "` a on `\e[` marca l'inici dels caràcters als quals s'aplica el color, i `\e[m` assenyala el final i `x;ym` és el codi de color:

Negro	0;30
Azul	0;34
Verde	0;32
Turquesa	0;36
Rojo	0;31
Morado	0;35
Marrón	0;33

La comanda per a canviar l'indicador a vermell fosc seria: `PS1="\e[0;31m[\u@\h\W]\$ \e[m "`

Si substituïm un 1 per un 0, obtindrem una versió més suau del mateix color. Podem experimentar amb aquests canvis temporalment fins que aconseguim l'indicador que desitgem. Si cometem un error, podem fer servir les tecles de fletxa per trobar una entrada en l'historial de comandes des de la qual podem canviar l'error o bé simplement tancar la finestra si estem treballant en un terminal virtual.

Quan trobem la fórmula del *prompt* que volem, obrim el fitxer `.bashrc` en un editor de text i substituïm la fórmula existent per la que hem ideat.

Si preferim, podem utilitzar el comandament `export`: per exemple, `export PS1="\e[0;31m[\u@\h\W]\$ \e[m "`

Ordres bàsiques

exit:

Finalitza la sessió de feina i torna a aparèixer el missatge de *login* per a que es connecti un altre usuari. És aconsellable desconnectar sempre per qüestions de seguretat. Els terminals no estan assignats de manera fixa a cada usuari, per tant podem entrar amb la mateixa identitat a terminals diferents. Si estem treballant amb un escriptori gràfic tenim 7 terminals diferents. Els 6 primers són textuais i el 7 és el gràfic. Per a entrar en un d'aquests terminals farem `CTRL + ALT + Fn` (a on `Fn` seran les tecles `F1` fins a `F7`).

who [am i]:

Ens informa dels usuaris que estan connectats actualment al sistema (1a columna), el terminal en el que es troben (2a columna) i la data i la hora a la que va entrar (3a columna).

Si fem `who am i` ens informa de nosaltres mateixos.

```
crisrina@1DAW:~$ who
linux      :0                2015-12-11 12:18 (:0)
linux      pts/2            2015-12-11 12:19 (:0)
linux      pts/17           2015-12-11 13:46 (:0)
armand     :1                2015-12-11 13:46 (:1)
armand     pts/35           2015-12-11 13:47 (:1)
crisrina   :2                2015-12-11 13:49 (:2)
crisrina   pts/38           2015-12-11 13:57 (:2)
crisrina@1DAW:~$
```

write:

Es fa servir per a comunicar-nos amb els altres usuaris que estan connectats al sistema en aquest moment. Escrivim el missatge i per a finalitzar-lo fem CTRL+D.

Exemple:

```
crisrina@1DAW:~$ write armand
Bon dia, com va?
```

```
armand@1DAW:~$
Message from crisrina@1DAW on pts/38 at 13:58 ...
Bon dia, com va?
EOF
```

mesg:

Activa o desactiva la recepció de missatges. Per a activar els missatges farem `$mesg y` i per a desactivar-los `$mesg n`

date:

Informa de l'hora i la data actuals. Fent servir `%` precedit de `+` podem donar format a la data amb els següents modificadors:

- `%d`: Dia
- `%m`: Mes
- `%y`: Any
- `%w`: Dia de la setmana.

Exemple:

```
armand@1DAW:~$ date +"Són les %r hores del %d de %m de %y"
Són les 02:05:30 hores del 11 de 12 de 15
```

```
armand@1DAW:~$ date
dv des 11 14:06:16 CET 2015
```

cal:

Ens mostra un calendari. La seva sintaxi és: `cal [mes] [any]`. Es pot fer servir de les següents cal

- `cal`: Ens mostra el calendari del mes i l'any actuals
- `cal any`: Ens mostra tot el calendari de l'any especificat.
- `cal mes any`: Ens mostra el calendari del mes i l'any especificat.

Exemple:

```
armand@1DAW:~$ cal
Desembre 2015
dg dl dt dc dj dv ds
          1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
armand@1DAW:~$ cal 11 2019
Novembre 2019
dg dl dt dc dj dv ds
          1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

uname:

Aquesta comanda es fa servir per a obtenir informació del sistema, tipus de màquina, sistema operatiu, tipus de processador, ... La seva sintaxi és `uname [-a] [-m] [-n] [--v] [-s] [-r]`. Veiem per a què serveix cada modificador i exemples del seu ús:

- `-a`: Mostra tota la informació
- `-m`: Ens dóna el tipus de hardware i processador.
- `-n`: Ens dóna el nom del host.
- `-v`: Ens dóna la informació de quan va ser instal·lat el sistema.
- `-s`: Ens dóna el nom del sistema.
- `-r`: Ens dóna la versió del nucli del sistema.

Exemple:

```
armand@1DAW:~$ uname -a
Linux 1DAW 4.2.0-16-generic #19-Ubuntu SMP Thu Oct 8 14:46:51 UTC 2015 i686 i686
i686 GNU/Linux
armand@1DAW:~$ uname -r
4.2.0-16-generic
armand@1DAW:~$ uname -s
Linux
armand@1DAW:~$ uname -v
#19-Ubuntu SMP Thu Oct 8 14:46:51 UTC 2015
armand@1DAW:~$ uname -m
i686
armand@1DAW:~$ uname -n
1DAW
```

passwd:

Ens permet modificar la nostra contrasenya. **ATENCIÓ:** En general no es pot fer servir qualsevol tipus de contrasenya. Depenent de la distribució podem tenir, per exemple, els següents requeriments: que la contrasenya sigui més llarga de 6 caràcters, diferent de l'usuari i de la darrera contrasenya.

clear:

Serveix per a netejar la pantalla. És similar a l'ordre `cls` de DOS.

script:

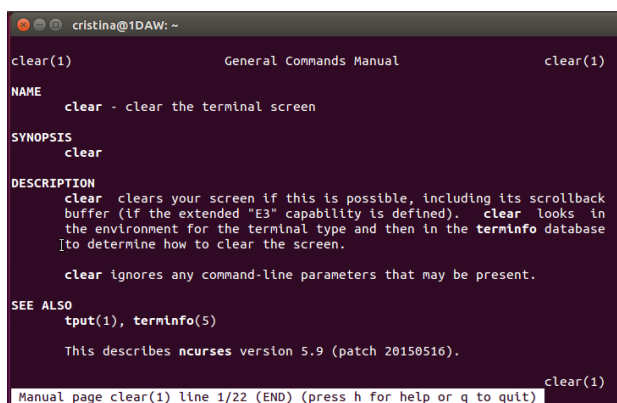
Aquesta comanda ens permet emmagatzemar en un arxiu tot el que l'usuari teclegi a partir del moment en que ho activem. És molt útil per a guardar una sessió de feina. Sintaxi: `script [-a] [nom_arxiu]`. Es fa servir de la següent manera:

- `script`: Emmagatzema tota la sessió en un arxiu anomenat `typescript`.
- `script nom_arxiu`: Emmagatzema tota la sessió en un arxiu anomenat `nom_arxiu`. Si aquest arxiu ja existeix l'esborra i crea un de nou.
- `script -a nom_arxiu`: Emmagatzema tota la sessió en un arxiu anomenat `nom_arxiu`. Si aquest arxiu ja existeix no l'esborra sinó que el completa.
- Per a finalitzar l'execució de `script` fem servir `exit`.

man:

Totes les ordres del SO es troben descrites al Manual del Programador de Linux. Aquest manual ens dóna la informació sobre totes les ordres de Linux i està dividit en les següents seccions:

1. Ordres i programes d'aplicació.
2. Trucades al sistema.
3. Subrutines.
4. Dispositius.
5. Format d'arxis.
6. Jocs.
7. Miscel·lània.
8. Manteniment.



La sintaxi de `man` és: `man [secció] ordre`

Si ens fixem a l'exemple veiem que hem executar `man clear` i ens ha donat tota la informació sobre la comanda `clear`. A més en diu `clear(1)` que vol dir que `clear` pertany a la secció 1. En general la informació sobre les comandes és molt extensa. Per moure'ns pel document farem servir `AvPag`, `RePag`, els cursors i per sortir farem `q`.

Si volem buscar dins d'aquests documents farem `/text_a_cercar`. És a dir, si volem cercar la paraula `terminal` farem `/terminal` i s'ens ressaltaran totes les aparicions d'aquesta paraula.