SISTEMAS OPERATIVOS
SESIÓN 1: INTÉRPRETE DE COMANDOS LINUX

Introducción

Linux es uno de los sistemas operativos más difundidos actualmente. Entre sus características principales se encuentran: es gratuito y de código abierto. Esta última significa que su código está disponible, lo que hace que sea utilizado en un ambiente didáctico puesto que se conoce muy bien su funcionamiento y estructura y el código está disponible y documentado para que sea más fácil modificarlo.
Linux incorpora una línea de comandos potente y unas herramientas de desarrollo evolucionadas.
Normalmente, el programador que trabaja en Linux no suele utilizar un entorno gráfico para desarrollar sino que usa la línea de comandos por su accesibilidad y versatilidad.
En esta asignatura se pretende que el estudiante adopte el perfil de programador sobre Linux y aprenda a desarrollar código sobre este sistema operativo utilizando los servicios y aplicaciones disponibles para facilitarle el trabajo.

Objetivos de la sesión

En esta sesión, el alumno se tiene que familiarizar con la línea de comandos de Linux. En concreto, tiene que aprender a:

- Ejecutar comandos de diversas formas
- Buscar ayuda sobre los comandos de sistema
- Redireccionar canales
- Comunicar comandos

Esta documentación quiere ser una guía de consulta rápida sobre el shell. Además, se incluyen ejemplos de lo explicado en el documento y se proponen ejercicios para que el alumno practique un poco los conceptos.

Intérprete de comandos

El intérprete de comandos o shell es una aplicación que facilita al usuario la ejecución de comandos. La comunicación entre usuario y sistema es a través de la línea de comandos. A través de esta línea de comandos, el usuario indica al shell que comandos quiere ejecutar y con qué parámetros. Un ejemplo podría ser:

```
> ls
```

El comando ls muestra el contenido de un directorio. Si no recibe ningún parámetro, muestra el contenido del directorio actual.
La mayoría de comandos de sistema pueden cambiar su comportamiento mediante parámetros y argumentos. Es responsabilidad del usuario saber cuales son y como funcionan. Por ejemplo, si usamos el parámetro –l con ls:

```
> ls -l
```

el resultado es diferente. La opción –l muestra más datos de los ficheros y directorios como el propietario, la longitud, los permisos que tiene, etc...

Pidiendo ayuda

El objetivo de este documento no es dar toda la información de los comandos que se pueden ejecutar junto con sus opciones. Para ellos, Linux incorpora una ayuda sobre todos ellos. Para invocar la ayuda, se utiliza:

```
> man COMANDO
```

donde COMANDO es el nombre del comando sobre el cual queremos ayuda. El formato de la ayuda es el siguiente:

```
> man ls

LS(1)                           FSF                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by
       default).  Sort entries alphabetically if none of -cftuSUX
       nor --sort.

       -a, --all
              do not hide entries starting with .

       -A, --almost-all
              do not list implied . and ..

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks

       -B, --ignore-backups
              do not list implied entries ending with ~

       -c     with  -lt:  sort  by, and show, ctime (time of last
              modification of file status information)  with  -l:
              show  ctime  and  sort  by  name otherwise: sort by
              ctime

       -C     list entries by columns

...
```

La información que nos muestra el comando man está estructurada en varias partes:

- El número que aparece en la primera línea indica la sección que estamos consultando.
- NAME muestra información escueta acerca de la palabra clave.
- SYNOPSYS presenta el interfaz. En el caso de un comando, son las opciones y parámetros. Para
- una llamada al sistema, son los parámetros, tipo del resultado, ficheros de cabecera...
- STANDARDS lista las reglas de estandarización que cumple el comando/llamada al sistema.
- OPTIONS indica la utilidad de las opciones del comando que aparece en SYNOPSYS
- PARAMETERS describe la interpretación de los parámetros de la llamada al sistema.
- DESCRIPTION detalla cuál es el funcionamiento del comando/llamada al sistema.
- EXIT VALUE enumera los posibles códigos de finalización del comando.
- RETURN VALUES indica los posibles valores de retorno de la llamada al sistema.
- ERRORS lista los posibles errores que se pueden producir al ejecutar la llamada al sistema.
- EXAMPLES muestra algunos ejemplos de utilització del comando.
- FILES indica los ficheros que están involucrados en la ejecución del comando.

Al invocar a man, aparece en la pantalla la primera página de la información solicitada. En este punto, el comando man espera que se apriete una de las siguientes teclas:

- ***barra espaciadora*** Pasa a la siguiente página de información
- ***b*** (backward) Pasa a la anterior página de información
- ***q*** (quit) Finalización del comando man
- ***return*** Se muestra la siguiente línea de información
- ***/palabra*** Busca una aparición de ***palabra*** en la información del comando.

La ayuda esta dividida en secciones:

- Sección 1: Comandos y programas de aplicación.
- Sección 2: Llamadas al sistema.
- Sección 3: Subrutinas y funciones de biblioteca.
- Sección 4: Formatos de ficheros.
- Sección 5: Miscelánea.
- Sección 6: Protocolos de red.
- Sección 7: Ficheros especiales del sistema.
- Sección 8: Procedimientos de mantenimiento y administración del sistema.
- Sección 9: Rutinas del núcleo (no estándar)
- Sección n: nuevo (obsoleto)
- Sección l: local (obsoleto)
- Sección p: público (obsoleto)
- Sección o: viejo (obsoleto)

Si queremos consultar la ayuda sobre un comando de una sección especifica, haremos:

```
> man SECCION COMANDO
```

Donde SECCION es un número de 1 a 8. Si no introducimos el número de sección, muestra la ayuda de la primera sección donde encuentre el comando. Por ejemplo, no es lo mismo hacer:

```
> man read
```

que

```
> man 2 read
```

La primera hace referencia al comando read ejecutable desde la línea de comandos. La segunda se refiere a la llamada al sistema read.

Obviamente, también se puede pedir ayuda sobre man:

```
> man man
```

Para una consulta más rápida por comando (lo que queremos buscar) podemos utilizar el comando whatis.

```
>whatis whatis
whatis (1)        - display manual page descriptions.
```

El formato del resultado de apropos es el siguiente:

COMANDO (SECCION) – DESCRIPCIÓN

Donde:

- COMANDO: es el nombre del comando a ejecutar
- SECCION: es la sección del man donde se encuentra la ayuda del comando.
- DESCRIPCIÓN: es una pequeña descripción de lo que hace el comando.

Como complemento a man y whatis, se utiliza apropos. Este comando busca una palabra (trozo de palabra) o frase dentro de una base de datos que contiene una pequeña descripción de todos los comandos disponibles en Linux y muestra por su salida las coincidencias. Apropos es muy útil cuando queremos buscar algún comando por su funcionalidad y no por su nombre.

```
>apropos apropos

apropos              (1)  - search the whatis database for strings
```

El resultado de apropos tiene el mismo formato que el de whatis.

Existe la posibilidad de pasar como argumento a apropos más de una palabra. En este caso, lo que hará será mostrar la lista de todas las coincidencias de todas las palabras sin repeticiones. Es decir, si hacemos:

```
> apropos apropos man
```

en realidad lo que hará será algo del estilo de:

```
>apropos apropos; apropos man
```

eliminado las repeticiones.

Es aconsejable hacer filtrados de las coincidencias para reducir la búsqueda. Imaginemos que solamente queremos mostrar las coincidencias en las que palabra1 y palabra2 aparezcan a la vez en el campo descripción. Para ello haremos:

```
> apropos palabra1 | grep palabra2
```

Esto se puede extender fácilmente a n palabras:

```
> apropos palabra1 | grep palabra2 | grep palabra3 ...
```

Formas de ejecución

En los anteriores ejemplos hemos visto diversas formas de ejecución de comandos. El usuario puede elegir entre diversas formas de ejecutar comandos. La primera de ellas es la ejecución normal. En ella, el usuario solamente indica que se ejecute un comando (o listas de comandos) en la línea de comandos. La línea de comandos tiene un aspecto como el siguiente:

```
> comando [parámetros] [argumentos] [redirección]
```

En esta clase de ejecución, el shell espera que el comando especificado acabe de ejecutarse para mostrar otra vez el prompt solicitando un nuevo comando.

Ejecución secuencial

También se puede indicar al shell que se quiere ejecutar una serie de comandos, pero que se quiere hacer de forma secuencial. Para ello, los comandos a ejecutar van separados por ; (punto y coma). Por ejemplo:

```
> comando1 ; comando2 ; …
```

Substitución de comandos

Hay veces que el resultado de un comando es necesario pasarlo como argumento a otro comando. Para ello, se tiene que sustituir en la línea de comandos el comando que crea los argumentos para otro comando, por los argumentos propiamente dichos. Para hacer esto, se utilizan las anticomillas simples:

```
> comando `comando1`
```

Por ejemplo: imaginemos que tenemos un fichero llamado pr que contiene una lista de nombres de directorios de los cuales queremos ver el contenido:

```
> cat pr
P1
P2
```

Para introducir esta lista en el comando ls, se tiene que hacer:

```
> ls `cat pr`
```

En este caso, el shell ejecuta primero cat pr, y sustituye, en la línea de comandos, este comando por su resultado quedado:

```
> ls P1 P2
```

Ejecución concurrente

El shell tiene una funcionalidad muy útil para ejecutar comandos cuyo tiempo de ejecución es muy largo. Consiste en la ejecución desatendida, o en background, de comandos. Esta forma de ejecución consiste, básicamente, en especificar al shell que no tiene que esperar la finalización de un comando para solicitar otra línea de comandos. Para hacer esto, el último carácter de la línea de comandos tiene que ser &. Por ejemplo:

```
> comando1 &
[1] 9214
```

En este caso, comando1 empieza a ejecutarse y, automáticamente, vuelve a salir el prompt solicitando un nuevo comando a ejecutarse. La línea [1] 9214 indica el número de proceso ejecutado en background, en este caso es el primero, y el identificador de proceso.

Cuando el proceso en background termine, se mostrará:

```
[1]  + done   comando1
```

donde [1] es el número de trabajo en background. Comando1 es el nombre del comando que se estaba ejecutando en background.

Pipes

Una funcionalidad interesante del sistema operativo es que se pueden comunicar entre ellos diversos comandos. En concreto, lo que se pretende es que un comando reciba como entrada estándar la salida estándar de otro comando. Para ello, el usuario tiene que especificar, mediante el símbolo | (pipe) que dos aplicaciones se comunican entre ellas. La línea de comandos podría ser:

```
> comando1 | comando2
```

En este caso, estamos especificando que el resultado de ejecutar comando1 (salida estándar) sea tratado como entrada estándar para el comando2. Es decir, el shell siempre conecta la salida estándar del comando situado a la izquierda de la pipe con la entrada estándar del comando que esta a la derecha.
El shell permite encadenar diversos comandos de esta forma. Por ejemplo, una línea de comandos como esta:

```
> comando1 | comando2 | comando3 | comando4
```

es perfectamente posible en Linux. En esta línea de comandos lo que estamos haciendo es comunicar la salida de comando1 con la entrada de comando2. A su vez, estamos conectando la salida de comando2 con la entrada de comando3 y lo mismo entre comando3 y comando4. Es necesario destacar que comunicaciones N a 1, 1 a N y N a N no es posible hacerlas.
En estos casos, el shell ejecuta todos los comandos a la vez, pero solamente espera a la finalización del último para continuar la ejecución.

Ejercicios:

1.- Consultar la ayuda sobre ls. En concreto, mirar las opciones –l, -a, -R.
2.- Consultar la ayuda sobre grep.
3.- Qué comando nos da información sobre el estado de los procesos actuales (current processes)?
4.- Por que hay diferencia en los resultados de apropos read y whatis read?

SISTEMA DE FITXERS

Podem considerar que un fitxer no és més que un conjunt de *bytes* que té associat un nom. Aquest nom permet que els usuaris puguin fer referència al fitxer de forma lògica, sense haver de preocupar-se de com s'emmagatzema el fitxer al disc; els serveis del sistema operatiu seran els responsables de les operacions de baix nivell.

El sistema de fitxers és la part del sistema operatiu encarregada de l'administració de les dades als dispositius d'emmagatzemament secundari. Proporciona mecanismes per emmagatzemar la informació de forma segura i

confidencial, tot i que un usuari ha de tenir la possibilitat de compartir els seus fitxers amb altres usuaris.

CARACTERÍSTIQUES DEL SISTEMA DE FITXERS DE UNIX

A continuació s'indiquen algunes de les característiques del sistema de fitxers Unix:

- Els fitxers estan organitzats en directoris. Un directori és un fitxer gestionat pel sistema operatiu que permet localitzar els fitxers que són al directori. A més, un directori pot contenir fitxers de tipus directori.
- El sistema de fitxers Unix té una estructura jeràrquica. Existeix un **directori arrel** (***root directory***) representat amb el símbol `/` que representa el punt d'inici d'aquesta estructura.
- Cada fitxer de tipus directori conté dos fitxers especials (`.` i `..`). El primer (.) fa referència al propi directori i el segon (`..`) fa referència al directori superior dins de la jerarquia de directoris.
- Dins d'aquesta estructura, cada usuari del sistema té assignat un directori on pot guardar els seus fitxers. Aquest directori és coneix com a **directori d'usuari** (***home directory***).
- Cada usuari pot associar proteccions als seus fitxers i directoris.
- El nom d'un fitxer pot tenir fins a 255 caràcters. Els fitxers on el nom comenci per `.` (punt) són **ocults**. Tot i que tots els caràcters són vàlids dins del nom d'un fitxer, és aconsellable que no apareguin metacaràcters (vegeu al final d'aquesta pàgina) del *shell* com `*`, `?`, `&`, `$`.

El sistema de fitxers de Unix us resultarà familiar ja que el de MsDOS i el de Windows estan inspirats en el de Unix. Una de les diferències més visibles és que el caràcter separador de directoris és diferent: a Unix és / mentre que a MsDOS i Windows és \.

Navegació al sistema de fitxers

Tot intèrpret de comandes té associat un **directori de treball** (o **actual** o ***current/working directory***) dins del sistema de fitxers. Quan inicieu un intèrpret de comandes, el seu directori de treball, per omissió, coincideix amb el directori d'usuari.

Per saber quin és el directori de treball heu d'executar la comanda **`pwd`** (*print working directory*), i es mostrarà el camí que hi ha des del directori arrel fins el directori de treball.

Per modificar el directori de treball associat a un intèrpret de comandes cal utilitzar la comanda **`cd`** (*change directory*). Per exemple, la comanda **`cd ..`** ens permet pujar un nivell dintre de l'estructura (a diferència de MsDOS, a Unix és precís separar la comanda `cd` i el directori `..` amb un espai en blanc); també permet anar directament a un directori (per exemple, **`cd /`** o **`cd /etc`**). Per tornar al directori d'usuari podeu executar la comanda **`cd`** sense paràmetres (executeu a continuació **`pwd`** per comprovar-ho).

COMANDES USUALS

Es presentaran algunes de les comandes més usuals de Unix amb exemples del seu ús. Consulteu el manual per veure més possibilitats de les comandes.

Creació/Destrucció de directoris

Les comandes que permeten crear i eliminar directoris són `mkdir` i `rmdir`. Totes dues esperen com a paràmetre el nom del directori a crear o a eliminar respectivament.

Metacaràcters per enumerar fitxers

L'intèrpret de comandes permet utilitzar metacaràcters per fer referència a varis fitxers de cop o a directoris d'ús comú.

- `*` : representa qualsevol seqüència de caràcters. `ls /bin/l*n` mostraria els fitxers del directori `/bin` amb nom que comença per `l` i acaba en `n`.
- `?` : representa un caràcter qualsevol. `ls /bin/l???n` mostraria els fitxers amb nom que comença per `l`, acaba en `n`, i té exactament 5 caràcters.
- `[ ]` : representa un caràcter dins d'un rang. `ls /bin/[aeiou]*[a-f]` mostraria els fitxers amb nom iniciat per vocal, i acabat en un caràcter entre `a` i `f`.
- `~` : representa el directori d'usuari[1]. ls ~ mostraria els fitxers del nostre directori d'usuari independentment de quin és el nostre directori actual.

Visualització del contingut d'un directori

La comanda `ls` mostra quins fitxers estan emmagatzemants a un directori. Per exemple, situeu-vos al directori arrel i executeu `ls`: es mostrarà la llista de fitxers del directori.

Noms relatius i absoluts

Sempre que invoqueu una comanda que necessiti com a paràmetre un nom de fitxer o directori, podeu escollir entre dos formes d'especificar el nom:

Nom absolut: camí des del directori arrel fins el fitxer (sempre començarà per `/` i el caràcter `/` separarà els directoris).

Nom relatiu: camí des del directori actual fins el fitxer (mai començarà per `/`).

Comandes bàsiques relacionades amb fitxers

Les comandes que realitzen les operacions més comuns sobre fitxers són:

- `rm` : esborra (*remove*) un fitxer. Té com a paràmetre el nom del fitxer a esborrar.

[1] Per escriure el caràcter `~` cal prémer AltGr 4.

`cp` : còpia (*copy*) un fitxer. Els seus paràmetres són el(s) nom(s) del(s) fitxer(s) a copiar i el directori destí, o el nom del fitxer a copiar i el nom que tindrà la còpia.

`mv`: canvia el nom (*move*) d'un fitxer. Cal indicar el nom antic i el nom nou.

`cat`: mostra el contingut d'un fitxer. Té com a paràmetre el nom del fitxer.

`more`: mostra el contingut d'un fitxer pàgina a pàgina. Cal indicar el nom del fitxer.

Exemple:

```
prompt$ cd                              <- Ens situem al directori d'usuari
prompt$ mkdir direc                     <- Creació d'un directori
prompt$ cd direc                        <- Ens situem al directori creat
prompt$  cp  /etc/passwd  .                <- Copiem /etc/passwd al directori
actual
prompt$ ls                              <- Llistem fitxers directori actual
passwd                                  <-   Apareix   la   còpia   del   fitxer
/etc/passwd
prompt$ mv passwd copia                 <- Canviem nom del fitxer passwd
prompt$ ls                              <- Tornem a llistar fitxers
copia
prompt$  cat  copia                      <- Ens mostra el contingut del fitxer
copia
root:x:0:0:root:/root:/bin/bash
...
prompt$ rm copia                        <- Esborrem fitxer copia
prompt$ ls                              <- Tornem a llistar fitxers
prompt$
```

Proteccions

Tots els usuaris d'un sistema Unix estan treballant sobre el mateix sistema de fitxers. Per garantir la confidencialitat de les dades apareix el concepte de proteccions dels fitxers. Les proteccions determinen les operacions que pot fer cada usuari sobre cada fitxer.

Es consideren tres tipus d'usuaris: l'usuari propietari del fitxer[2], els usuaris que pertanyen al seu mateix grup d'usuaris[3] i la resta d'usuaris de la màquina.

Es consideren tres tipus d'operacions: lectura (r), escriptura (w) i execució (x)

[2] Al fitxer `/etc/passwd` podeu veure tots els usuaris del sistema. La primera paraula de cada línia (és a dir, fins al caràcter `:`) és l'identificador d'un usuari del sistema. La resta de la línia conté dades de l'usuari com el seu directori d'usuari, el *shell* que té associat

[3] El fitxer `/etc/group` conté informació sobre els grups d'usuaris definits al sistema.

Les proteccions d'un fitxer es representen amb nou caràcters. Els tres primers indiquen les operacions que pot realitzar el propietari del fitxer, els tres següents fan referència a les operacions realitzables pels usuaris del mateix grup i els tres darrers fan referència a les operacions realitzables per la resta d'usuaris de la màquina. Per exemple, les proteccions rwxr----- indiquen que el propietari pot fer totes les operacions sobre el fitxer (rwx), que els membres del seu grup només poden fer lectures (r--) i que la resta d'usuaris no pot fer cap operació sobre el fitxer (---).

Per modificar les proteccions d'un fitxer s'utilitza la comanda chmod, que codifica les proteccions seguint el codi r=4, w=2 i x=1 i sumant els drets de cada tipus d'usuari[4]. És a dir, les proteccions rwxr-x--- es representen com 750 (4+2+1, 4+0+1, 0+0+0). Per exemple, chmod 631 file fa que file passi a tenir proteccions rw--wx--x.

Directoris habituals

Presentem els directoris més habituals de tot sistema de fitxers Unix i el seu contingut:

- `/bin` i `/usr/bin`: executables de moltes comandes (`ls`, `cp`,...) i aplicacions
- `/dev` : fitxers que representen els dispositius del sistema (terminals,...)
- `/etc` : dades de configuració (fitxer de *passwords* `/etc/passwd`, ...)
- `/home` : directoris d'usuari dels usuaris de la màquina
- `/proc`: informació sobre els programes en execució a la màquina
- `/tmp` : arxius temporals dels processos
- `/usr/include` : fitxers de capçalera de les crides al sistema
- `/usr/lib` : llibreries de funcions

FILTRES

Els filtres són comandes que acostumen a rebre dades per l'entrada estàndar, processar-les i escriure el resultat per la seva sortida estàndar. Els filtres acostumen a estar comunicats amb altres comandes mitjançant *pipes*. Es presenten exemples d'ús d'aquests filtres; és aconsellable executar-los pas a pas per veure l'efecte de cada filtre.

more

Mostra per pàgines les dades rebudes per l'entrada estàndar. Prement la barra d'espai podeu avançar una pàgina i prement `q` podeu finalitzar l'execució de `more`. Exemple:

```
prompt$ ls -l /bin | more
```

[4]També podeu pensar en la codificació en base 2 de les proteccions: `r`=100, `w`=010 i `x`=001. Com a exemple, les proteccions `rwxr-x---` es representarien com a 111101000.

sort

Ordena les línies d'un fitxer d'entrada segons el criteri que indiquem. Podem ordenar alfabèticament o numèricament, ascendentment o descendentment, definir la part de la línia que s'ha de considerar per ordenar,... Exemples:

`prompt$` **`sort /etc/passwd`** Mostra per *stdout* el resultat d'ordenar alfabèticament el fitxer `/etc/passwd` considerant el contingut sencer de la línia.

`prompt$` **`sort -t: -k3 -n /etc/passwd`** Mostra el resultat d'ordenar numèricament (paràmetre `-n`) pel tercer camp (paràmetre `-k3`), considerant que el caràcter separador de camps és `:` (paràmetre `-t:`). Proveu-la també sense el `-n`.

`prompt$` **`ls -l /etc | sort -k5 -n -r | more`** Ordena el resultat de `ls -l /etc` pel cinquè camp (la mida del fitxer) descendentment (paràmetre `-r`). Fem servir el separador de camps per omissió (un o més espais en blanc).

cut

Selecciona una porció de totes les línies d'un fitxer d'entrada. Podem especificar la porció de la línia com a un rang de caràcters o com a un rang de camps. Exemples:

`prompt$` **`cut -d: -f3,5 /etc/passwd`** Mostra el tercer i el cinquè camp de totes les línies de `/etc/passwd` considerant que el separador de camps és `:` (paràmetre `-d`). Fixeu-vos que l'especificació del separador difereix respecte `sort`.

`prompt$` **`date | cut -c12-19`** Mostra únicament l'hora actual (estem seleccionant el rang de caràcters entre la columna 12 i la 19).

head

Selecciona les primeres línies o els primers caràcters d'un fitxer. Exemple:

`prompt$` **`sort /etc/passwd | head -2 | cut -d: -f1`** Mostra els dos primers identificadors d'usuari (alfabèticament parlant) de la màquina.

tail

Selecciona les darreres línies o els darrers caràcters d'un fitxer. Exemple:

`prompt$` **`ls -l /etc | sort -k5 -n | tail -2`** Mostra els dos fitxers més grans de `/etc`.

tr

Fa una traducció caràcter a caràcter de l'entrada estàndar. Exemples:

`prompt$` **`tr aeiou AEIOU </etc/passwd`** Mostra el resultat de substituir les vocals minúscules per majúscules de `/etc/passwd`. El primer paràmetre (`aeiou`) són els caràcters a substituir; el segon per quins substituir-los (correspondència u a u).

`prompt$` **`ls -l / | tr -s ' '`** Mostra el resultat de `ls -l` separant els diferents camps per un únic espai en blanc (la comanda `tr` està

substituint totes les aparicions consecutives de l'espai en blanc per un únic espai en blanc).

grep

Mostra les línies que continguin una o vàries cadenes de caràcters a unes posicions determinades. La comanda `grep` disposa d'un seguit de metacaràcters per a especificar les cadenes de caràcters; cal no confondre'ls amb els del *shell*. Exemples:

`prompt$` **`grep ro /etc/passwd`** Mostra les línies de `/etc/passwd` que contenen `ro` a qualsevol posició

`prompt$` **`ls -l /etc | grep ^d`** Mostra les línies retornades per `ls` que comencen per `d` (metacaràcter `^`), és a dir, mostra els directoris de `/etc`.

`prompt$` **`ls -l /etc | grep ^-.....r-x`** Mostra les línies retornades per ls que comencen per `-`, continuen amb 6 caràcters qualsevol (metacaràcter `.`) i continuen amb els caràcters `r-x`.

`prompt$` **`grep bash$ /etc/passwd`** Mostra les línies del fitxer `/etc/passwd` que acaben amb `bash` (metacaràcter `$`).

`prompt$` **`grep -v /bin/bash /etc/passwd`** Mostra les línies del fitxer `/etc/passwd` que no contenen (paràmetre `-v`) la paraula `/bin/bash`.

`prompt$` **`grep ^n[aeiou] /etc/passwd`** Mostra les línies de `/etc/passwd` que comencen per `na`, `ne`, `ni`, `no` o `nu` (els metacaràcters `[]` especifiquen rangs).

wc

Compta el nombre de línies, paraules i caràcters d'un fitxer. Exemples:

`prompt$` **`wc -c /etc/passwd`** Mostra el nombre de caràcters (paràmetre `-c`) de `/etc/passwd`.

`prompt$` **`ls -l / | grep ^d | wc -l`** Mostra el nombre de línies (paràmetre `-l`) retornades per `ls -l /|grep ^d`, que és el nombre de directoris de `/`.

uniq

Compacta en una línia totes les línies repetides i consecutives. Exemple:

`prompt$` **`ps -e | cut -c25- | sort | uniq -c`** Per a cada comanda en execució, mostra el nombre de processos que l'estan executant