
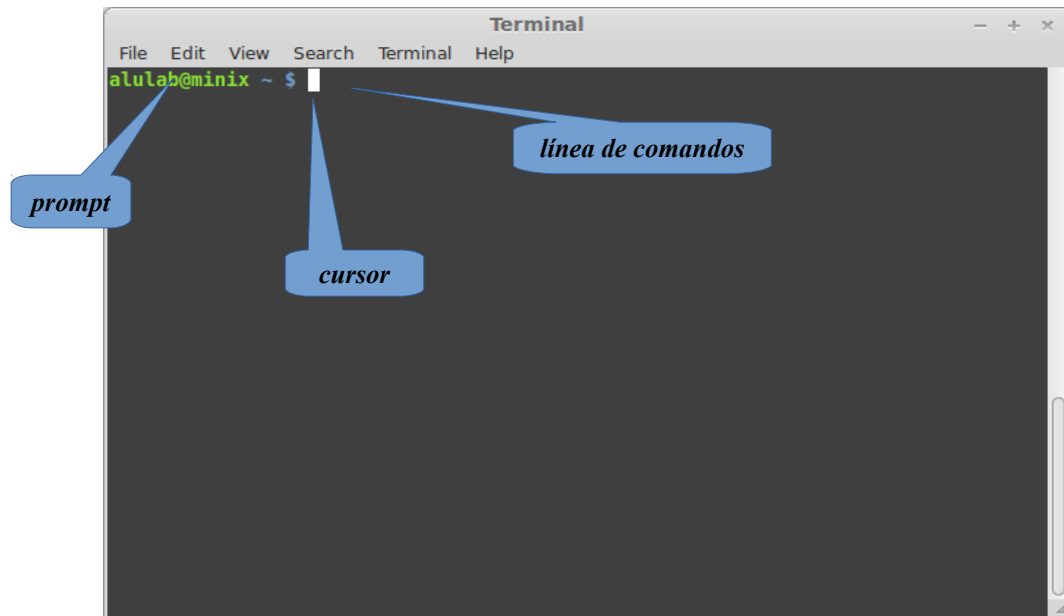


PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FACULTAD DE CIENCIAS E INGENIERIA
INGENIERIA INFORMATICA
INF239 SISTEMAS OPERATIVOS

Laboratorio Preliminar 0A

En este primer laboratorio todos los ejercicios se llevará a cabo en una terminal virtual (). Presione Ctrl + Alt + t para obtener la *Terminal*.



Para ejecutar cualquier comando, primero debe de escribirlo en la terminal y luego presionar la tecla <Enter>. Muchos comando permiten la especificación de opciones al momento de ejecutarlo. Estas opciones normalmente van precedidas por un guión (-) seguidas de una letra. Por ejemplo en el comando `ls -l`, la opción '-l' indica que debe mostrarse el listado en un formato largo.

Usted puede obtener una explicación más detallada con el manual de ayuda en línea: **man**. El manual en línea es de gran utilidad durante una sesión, si desea información acerca de algún comando, puede invocarlo de la siguiente forma: `man <nombre_comando>`. Para salir del manual de ayuda en cualquier momento presione la tecla **q**. Haciendo uso del manual, resuelva los ejercicios propuestos en cada caso.

A continuación algunos comandos básicos

Mostrando la fecha y hora. El comando *date*

```
alulab@minix ~ $ date
mar mar 12 18:36:29 PET 2013
alulab@minix ~ $
```

`date` imprime el día de la semana, mes, día, hora (reloj de 24 horas, la zona de tiempo del sistema) y el año.

Ejercicios

- 1.- Haciendo uso del comando `date` muestre la fecha de su cumpleaños. Si su cumpleaños es el 27 de Octubre, al comando `date` se le debe pasar "27 Oct", mediante la opción apropiada .
- 2.- Muestre por pantalla la hora UTC (Tiempo Universal Coordinado).

Mostrando el calendario. El comando `cal`

```
alulab@minix ~ $ cal
      Marzo 2013
Su Mo Tu We Th Fr Sa
 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
alulab@minix ~ $
```

Ejercicios. Muestre por pantalla:

- 1.- El calendario de todos los meses del presente año.
- 2.- Los meses transcurridos desde marzo del 2012 hasta marzo del 2013

Encontrando quién ha iniciado sesión. El comando `who`

```
alulab@minix ~ $ who
alejandro tty7      2013-03-12 12:38 (:0)
alejandro pts/0     2013-03-12 18:44 (:0)
alulab pts/2        2013-03-12 18:14
alulab@minix ~ $
```

En este ejemplo, dos usuarios con el mismo nombre han iniciado sesión. En la computadora en la que usted ejecute el comando la salida podría ser distinta. Con cada nombre de usuario se muestra el número de (terminal) *tty* o *pts* (pseudo terminal) de ese usuario, y el día y la hora en que el usuario ha iniciado sesión. El número de *tty/pts* es un único número de identificación que el sistema otorga a cada terminal (pseudo terminal o dispositivo de red) en el que el usuario ha iniciado sesión.

Ejercicios. Muestre por pantalla:

- 1.- Las cabeceras de cada columna que el comando `who` presenta en la pantalla.
- 2.- Sólo los nombres y la cantidad de usuarios que están conectados.

Haciendo eco de caracteres. El comando `echo`

El comando `echo` imprime (o hace eco) en la terminal cualquier cosa que escribe en la línea (hay algunas excepciones que se mencionarán después):

```
alulab@minix ~ $ echo este es mi primer lab preliminar de S0
este es mi primer lab preliminar de S0
alulab@minix ~ $ echo
alulab@minix ~ $ echo uno      dos      tres
uno dos tres
alulab@minix ~ $
```

En el último ejemplo se puede observar que `echo` extrae los espacios en blanco adicionales entre palabras. Eso es porque en un sistema *Unix like*, las palabras son importantes; los espacios en blanco están meramente allí para separar las palabras, generalmente el sistema ignora los espacios en blanco extras.

Ejercicios

- 1.- Modifique el último ejemplo de forma que `echo` no ignore los espacios en blanco.
- 2.- Haciendo uso del comando `echo` muestre por pantalla *hola como estas*, donde cada palabra debe estar en filas distintas pero en la misma columna.

Trabajando con directorios

El directorio en el que se encuentra después de haberse identificado, se denomina *home*. En este directorio usted tiene derecho de crear, copiar, borrar, renombrar o mover directorios y archivos. Si usted en algún momento necesita saber cuál es su *home*, puede imprimir la variable de entorno `HOME`, con la siguiente orden en la línea de comando:

```
alulab@minix ~ $ echo $HOME
/home/alulab
alulab@minix ~ $
```

Para crear directorios utilizamos la orden `mkdir`. Por ejemplo si deseamos crear tres directorio en nuestro *home*, utilizaremos la siguiente línea.

```
alulab@minix ~ $ ls
Descargas  Escritorio  milista     Música      Público
Documentos Imágenes   milista.old Plantillas  Vídeos
alulab@minix ~ $ mkdir dir1 dir2 dir3
alulab@minix ~ $ ls
Descargas  dir2  Documentos  Imágenes  milista.old  Plantillas  Vídeos
dir1      dir3  Escritorio  milista   Música        Público
alulab@minix ~ $
```

En este caso no se obtiene mensaje alguna tras ejecutarse la orden. Una de las formas de ver si se han creado los directorios, es listando su contenido. Esto lo logramos con el comando `ls`, que después será explicado con más detalle.

Se puede crear subdirectorios intermedios en una sola orden, sin cambiar de directorio.

```
alulab@minix ~ $ mkdir -p dir1/dir1.1/dir1.1.1
alulab@minix ~ $
```

Crearé el directorio `dir1.1.1` después de haber creado `dir1.1` y por debajo de este.

Se denomina directorio actual, o directorio de trabajo al directorio en que se encuentra en el momento en que se ejecuta una orden. Al iniciar una sesión el directorio actual es su *home*. Para saber cuál es el directorio de trabajo use la orden `pwd`.

```
alulab@minix ~ $ pwd
/home/alulab
alulab@minix ~ $
```

Para cambiar de un directorio a otro, empleamos la orden `cd`. Por ejemplo para ingresar al directorio de mayor profundidad desde el *home*.

```
alulab@minix ~ $ cd dir1/dir1.1/dir1.1.1/
alulab@minix ~/dir1/dir1.1/dir1.1.1 $ pwd
/home/alulab/dir1/dir1.1/dir1.1.1
alulab@minix ~/dir1/dir1.1/dir1.1.1 $
```

Para regresar al directorio *home* desde cualquier directorio no es necesario indicar toda la ruta, simplemente indicar la siguiente orden: `cd`. Para regresar al directorio previo donde nos encontramos podemos hacerlo del siguiente modo: `cd -`

```
alulab@minix ~/dir1/dir1.1/dir1.1.1 $ cd
alulab@minix ~ $ pwd
/home/alulab
alulab@minix ~ $ cd -
/home/alulab/dir1/dir1.1/dir1.1.1
alulab@minix ~/dir1/dir1.1/dir1.1.1 $ pwd
/home/alulab/dir1/dir1.1/dir1.1.1
alulab@minix ~/dir1/dir1.1/dir1.1.1 $
```

Como puede verse, `cd` – puede usarse para intercambiar entre dos directorios.

En los ejemplos arriba mostrados se han indicado rutas tales como `dir1/dir1.1/dir1.1.1` y `/home/alulab/dir1`. Cuando la ruta inicia con un *slash* (/) se dice que es una **ruta absoluta**, e indica que la ruta debe iniciarse desde el directorio raíz. En caso contrario se denomina **ruta relativa** y se inicia en el directorio actual. Se denomina directorio padre al directorio que se encuentra por encima del directorio actual. Tanto el directorio actual como el directorio padre pueden ser representados por dos símbolos:

. directorio actual
.. directorio padre

Estos símbolos pueden ser usados en ordenes como `cd`. Por ejemplo para regresar al directorio padre.

```
alulab@minix ~/dir1/dir1.1/dir1.1.1 $ pwd
/home/alulab/dir1/dir1.1/dir1.1.1
alulab@minix ~/dir1/dir1.1/dir1.1.1 $ cd ..
alulab@minix ~/dir1/dir1.1 $ pwd
/home/alulab/dir1/dir1.1
alulab@minix ~/dir1/dir1.1 $
```

Debe respetar el espacio entre `cd` y el punto-punto (`..`)

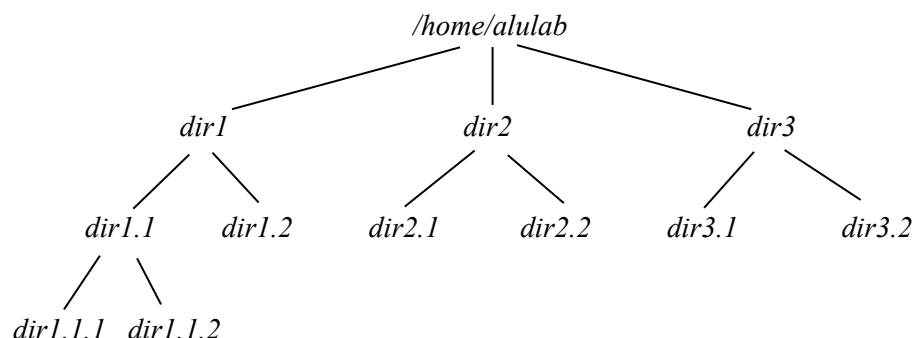
Es importante mencionar que el símbolo `~` representa al *home* en este caso en particular el directorio `/home/alulab`. Esto puede abreviar la forma de nombrar las rutas, por ejemplo si se desea hacer referencia al directorio `/home/alulab/dir1/dir1.1` se puede emplear `~/dir1/dir1.1`

Para borrar directorio vacíos empleamos el comando `rmdir`.

```
alulab@minix ~ $ ls
Descargas  Documentos  Imágenes  Plantillas  Vídeos
dir1       Escritorio  Música    Público
alulab@minix ~ $ mkdir fantasma
alulab@minix ~ $ ls
Descargas  Documentos  fantasma  Música    Público
dir1       Escritorio  Imágenes  Plantillas  Vídeos
alulab@minix ~ $ rmdir fantasma
alulab@minix ~ $ ls
Descargas  Documentos  Imágenes  Plantillas  Vídeos
dir1       Escritorio  Música    Público
alulab@minix ~ $
```

Ejercicio

1.- Ejecute las ordenes necesaria para obtener el siguiente árbol de directorios:



2.- Haciendo uso del comando `tree` compruebe que el árbol que ha creado en el ejercicio anterior, es el correcto.

3.- ¿Cuántas ejecuciones del comando `rmdir`, como mínimo, son necesarias para borrar todo el árbol de directorios creados en el ejercicio anterior? Borre estos directorios (*dir**)

Trabajando con archivos

El sistema reconoce sólo tres tipos básicos de archivos: archivos ordinarios, archivos de directorios, y archivos especiales. Un archivo ordinario es sólo eso: cualquier archivo en el sistema que contiene datos, texto, instrucciones de un programa, o cualquier otra cosa. Los directorios no son sino archivos cuya estructura está definida de acuerdo al sistema operativo. Como se puede deducir de su nombre, un archivo especial tiene un significado especial para el sistema y típicamente está asociado con alguna forma de entrada/salida. Un nombre de archivo puede estar compuesto por cualquier carácter disponible desde el teclado (y aún los que no lo están) y el número de caracteres que contiene el nombre no debe ser mayor de 255 en caso contrario los caracteres extras son ignorados.

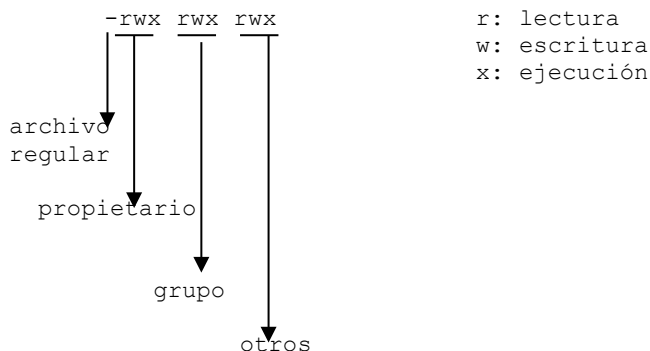
Listando los archivos. El comando `ls`

```
alulab@minix ~ $ ls
Descargas  Escritorio  Música     Público
Documentos Imágenes   Plantillas Vídeos
alulab@minix ~ $ ls -l
total 32
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Descargas
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Documentos
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Escritorio
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Imágenes
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Música
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Plantillas
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Público
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Vídeos
alulab@minix ~ $
```

El número a continuación de la palabra `total`, al inicio de la lista en el segundo ejemplo, no corresponde al número de archivos como se podría pensar, sino al número total de KB que se ha asignado a los archivos listados en el directorio (no confundir con el tamaño del archivo). El sistema asigna zonas para almacenar los archivos, cada zona es de 4KB. La asignación de zonas se hace redondeando al mayor entero el resultado de dividir el tamaño del archivo entre bloques de 4KB. Por ejemplo si el tamaño de un archivo es de 179 bytes, Linux le asigna una zona, es decir 4KB. Si el archivo es de 4097 bytes el sistema le asigna 2 zonas de 4KB cada una. Esta información sólo se muestra sólo cuando se lista todo el contenido de un directorio.

El **propietario** de un archivo es por lo general el usuario que lo ha creado. En el listado anterior la tercera columna indica que el propietario es el usuario `ladmin`. Todo usuario debe ser miembro de un **grupo**. En nuestro ejemplo la cuarta columna, indica que pertenece al grupo `ladmin`. Cualquier otro usuario que no pertenezca al grupo y que no sea el propietario se les denomina comúnmente **otros**.

Cuando en un listado largo de archivos (`ls -l`) aparece un guión en la primera columna, indica que se trata de un archivo regular. Las siguientes tres letras: `rwX` indican que el propietario tiene permiso de leer, escribir y ejecutar si se trata de un archivo. Cuando un permiso le está negado, en lugar de alguna de las tres letras (`r,w,x`) se encuentra un guión. Las siguientes tres letras tienen el mismo significado pero corresponden a los permisos que tienen los miembros del grupo. Y las últimas tres letras corresponden a los permisos que se les otorga sobre el archivo a otros usuarios que no son ni el propietario ni miembro del grupo.



Por ejemplo -rwxrw-r-- 1 alulab alulab 708 Mar 17 12:30 Makefile

El primer guión indica que se trata de un archivo regular. El propietario *alulab* puede leer, escribir y ejecutar este archivo (siempre y cuando sea un archivo apropiado). Cualquier usuario que sea miembro del grupo *alulab* puede leer y escribir sobre él. Pero cualquier otro usuario sólo podrán leerlo.

Ejercicios

- 1.- Cuando se hace un listado largo (opción *l*) la opción por defecto es listar el contenido del directorio ordenado alfabéticamente. Obtenga un listado pero en orden inverso.
- 2.- Si se desea listar en formato largo información del directorio */etc*, un primer intento sería el siguiente comando: `ls -l /etc`, sin embargo no se obtendría el resultado deseado, porque este mostraría información acerca de cada archivo que contiene el directorio */etc*. Escriba en la terminal el comando `ls` con la opción apropiada para mostrar sólo información del directorio */etc* en lugar de su contenido.
- 3.- Muestre el contenido del directorio */etc* en formato largo ordenando los archivos basandose en la última fecha de modificación de cada archivo.
- 4.- Lo mismo que el caso anterior, pero en orden inverso.
- 5.- Se desea mostrar los archivos contenidos en */usr/bin*, ordenados por tamaño.
- 6.- Liste en formato largo el contenido del directorio */etc* de forma que también se muestre el contenido de todos los directorios que se encuentran en */etc*. Si estos a su vez contienen directorios, también deben mostrarse.
- 7.- Busque información en Internet acerca de ¿qué es un inodo (o nodo-i)? Haciendo uso del comando `ls` obtenga un listado de los archivos contenidos en */usr/bin*, con sus respectivos inodos.

Mostrando el estado de un archivo. El comando *stat*

```
alulab@minix ~ $ stat /etc/passwd
  File: '/etc/passwd'
  Size: 1739          Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d   Inode: 531859       Links: 1
Access: (0644/-rw-r--r--)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2013-03-12 18:14:28.514054154 -0500
Modify: 2013-03-12 18:14:28.458053878 -0500
Change: 2013-03-12 18:14:28.478053966 -0500
 Birth: -
alulab@minix ~ $ stat /usr/bin/haddock
  File: '/usr/bin/haddock'
  Size: 35446584      Blocks: 69232      IO Block: 4096   regular file
Device: 806h/2054d   Inode: 5646234      Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2013-03-08 19:25:18.199680092 -0500
Modify: 2012-09-04 13:07:06.000000000 -0500
Change: 2013-03-08 19:11:43.071638086 -0500
 Birth: -
alulab@minix ~ $
```

Ejercicios

- 1.- Haciendo uso del comando `stat` obtenga por pantalla las siguientes salidas:
 - a) Nombre de archivo: */usr/bin/haddock*
Tipo de archivo: regular file
 - b) Número de bloques lógicos asignados: 69232
Tamaño de cada bloque lógico: 512

Concatenando archivos. El comando *cat*

```
alulab@minix ~ $ cat /usr/share/applications/swi-prolog.desktop
[Desktop Entry]
Name=SWI-Prolog
Comment=Prolog Interpret
Exec=swi-prolog
Icon=/usr/share/pixmaps/swi-prolog.png
Terminal=true
Type=Application
Categories=Development;
StartupNotify=true
NoDisplay=false
alulab@minix ~ $
```

Concatena varios archivos y

los muestra por la salida estándar. Si solo se proporciona el nombre de un solo archivo, el contenido de este es mostrado por la salida estándar. Los archivos tipo texto (aquellos que contienen caracteres imprimibles) se mostrarán tal cual, mientras que para los archivos binarios, se mostrará caracteres extraños.

Ejercicios

- 1.- Obtenga la salida del ejemplo anterior, pero ahora mostrando un número de línea en el lado izquierdo.
- 2.- Muestre el contenido de todos los archivos con extensión *.sh* que se encuentran en */usr/bin*

Contando el número de palabras de un archivo. El comando *wc*

```
alulab@minix ~ $ wc /usr/share/applications/swi-prolog.desktop
10 12 202 /usr/share/applications/swi-prolog.desktop
alulab@minix ~ $
```

El comando *wc* lista tres números seguidos por el nombre. El primer número representa el número de líneas, el segundo es el número de palabras y el tercero es el número de caracteres contenidos en el archivo.

Ejercicios

- 1.- Muestre por pantalla solo el número de líneas del archivo */usr/local/bin/squeak.sh*
- 2.- También muestre la longitud de la línea más larga, del mismo archivo.

Entrada, salida y error estándar

Cada vez que el *shell* ejecuta un programa, prepara para éste los dispositivos de entrada, salida y error estándar. Por defecto el dispositivo asociado a la entrada estándar es el teclado y para la salida y error estándar es la pantalla. De esta forma el programa ejecutable tiene un medio de comunicación con el mundo exterior. El dispositivo correspondiente al teclado y a la pantalla se llama */dev/tty1*, el de la impresora es */dev/lp*, el del puerto serial es */dev/tty00*. También existe un dispositivo ficticio de salida que recibe los datos y los desecha. Este dispositivo se llama */dev/null*. Muchas veces se desea que los resultados de un programa sean guardados en un archivo en lugar de imprimirlos por pantalla, otras veces se desea ingresar los datos desde un archivo en lugar de ingresarlos desde el teclado. Todo ello por supuesto sin tener que modificar el programa. El *shell* acepta algunos símbolos que permiten cambiar los dispositivos estándar para un programa en ejecución, por ejemplo, con el siguiente comando:

```
alulab@minix ~ $ ls -l /etc > milista
alulab@minix ~ $
```


Se redirecciona el dispositivo de salida estándar para el comando `ls`, debido al símbolo “>”, hacia el archivo `milista` en el directorio de trabajo. Por consiguiente los resultados del comando `ls` en vez de aparecer por pantalla serán grabados en un archivo creado en ese momento, si ya existiese se sobre escribe. Para que la salida sea añadida en lugar de sobre escribirla al archivo existente, use el símbolo “>>” colocando a continuación el nombre del archivo.

El símbolo “<” redirige la entrada estándar, por ejemplo

```
alulab@minix ~ $ wc -l < milista
247
alulab@minix ~ $
```

presenta por pantalla la cantidad de líneas que contiene el archivo `milista`

El símbolo de canalización “|” crea una tubería (*pipe*) entre varios programas que se ejecutan en forma concurrente. Veamos que es lo que hace el siguiente comando

```
alulab@minix ~ $ du -a /usr/bin | sort -nr | more
257676 /usr/bin
34616 /usr/bin/haddock
19688 /usr/bin/gnat-gps
9592 /usr/bin/cabal
8448 /usr/bin/net.samba3
7516 /usr/bin/rpcclient
6260 /usr/bin/smbget
6060 /usr/bin/smbclient
6044 /usr/bin/smbpasswd
6028 /usr/bin/smbcacs
6020 /usr/bin/smbcquotas
5960 /usr/bin/smbtree
5532 /usr/bin/gimp-2.8
4736 /usr/bin/Xvnc4
4564 /usr/bin/gdb
4424 /usr/bin/aptitude-curses
3336 /usr/bin/smbpool
3000 /usr/bin/mono
2804 /usr/bin/mplayer
2800 /usr/bin/python3.2mu
2596 /usr/bin/python2.7
2560 /usr/bin/gimp-console-2.8
2348 /usr/bin/vim.gnome
2212 /usr/bin/Xorg
2152 /usr/bin/alex
2076 /usr/bin/Xephyr
2044 /usr/bin/pdbedit
1972 /usr/bin/vim.basic
1884 /usr/bin/eventlogadm
1836 /usr/bin/happy
1808 /usr/bin/ld.gold
1608 /usr/bin/nmblookup.samba3
1592 /usr/bin/smbstatus.samba3
1552 /usr/bin/smbcontrol
1552 /usr/bin/profiles
1532 /usr/bin/testparm.samba3
--More--
```

La salida del comando `du` (*disk usage*), muestra el espacio ocupado por cada archivo (en sectores) contenido en el directorio especificado (directorio de trabajo por defecto). Esta salida se pasa como entrada para el comando `sort` quien ordena esta lista por tamaños (primera columna de la salida del comando `du`) de mayor a menor (considera la columna como números). Este a su vez lo pasa a `more` para hacer una pausa en caso de que sea necesario.

Puede limpiar su pantalla con el comando `clear`.

Ejercicios

- 1.- Haciendo uso de entubamiento muestre ¿cuántos archivos con extensión `.sh` existen en `/usr/bin`?
- 2.- Concatene todos los archivos hallados en el ejercicio anterior en uno solo con nombre `scripts.sh` en el directorio actual.
- 3.- Haciendo uso de entubamiento muestre ¿cuántos archivos (a los que puede acceder) con extensión `.c` existen en el disco duro. El error estándar redirigirlo a `/dev/null`

Identificando el tipo de archivo. El comando `file`

```
alulab@minix ~ $ file /usr/bin/yes
/usr/bin/yes: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU
/Linux 2.6.24, BuildID[sha1]=0x56f22f145c96438dd4b506c2b31494658d04196b, stripped
alulab@minix ~ $ file /usr/local/bin/squeak
/usr/local/bin/squeak: POSIX shell script, ASCII text executable
alulab@minix ~ $ file /usr/share/pixmaps/swi-prolog.png
/usr/share/pixmaps/swi-prolog.png: PNG image data, 64 x 64, 16-bit/color RGBA, non-interlaced
alulab@minix ~ $ file /usr/share/pixmaps/haskell.jpg
/usr/share/pixmaps/haskell.jpg: JPEG image data, JFIF standard 1.01
alulab@minix ~ $
```

Ejercicios

- 1.- Todos los ejemplos del comando `file` se pudieron escribir en una sola línea. ¿Cuál es esta?
- 2.- Empleando el comando `file` muestre información de cada una de las particiones del disco duro.

Haciendo una copia de un archivo. El comando `cp`

Si se desea copiar archivos, entonces deberá emplear el comando `cp`. Por ejemplo, para realizar una copia del archivo `milista` con el nombre `milista.old`

```
alulab@minix ~ $ ls
Descargas  Escritorio  milista  Plantillas  Vídeos
Documentos Imágenes   Música   Público
alulab@minix ~ $ cp milista milista.old
alulab@minix ~ $ ls
Descargas  Escritorio  milista  Música  Público
Documentos Imágenes   milista.old  Plantillas  Vídeos
alulab@minix ~ $
```

El formato del comando `cp` con algunas de sus opciones se indica a continuación::

```
cp [-fip] fuente destino
cp [-fiprR] fuente1 fuente2 . . . dir_destino
```

La primera forma se aplica cuando el destino es un archivo, en este caso el origen debe ser también un archivo. La segunda forma se aplica cuando el destino es un directorio y el origen puede ser un archivo o conjunto de archivos, indicado inclusive por comodines.

En cuanto a las opciones podemos mencionar las siguientes:

- i Pregunta de modo iterativo si el archivo se desea sobrescribir, obviamente cuando el archivo que se va a copiar ya existe, en caso contrario no tiene efecto.
- r -R Se copian los archivos y los subdirectorios recursivamente contenidos en el directorio fuente. En este caso el destino tiene que ser un directorio.

Ejercicios

- 1.- Copie a su *home* todos los archivos con extensión *.h* que se encuentran en */usr/include*
- 2.- Copie el contenido completo del directorio */etc* dentro del directorio *Descargas* de su *home*.

Renombrando un archivo. El comando *mv*

Un archivo puede ser renombrado con el comando *mv* y a continuación el nombre actual seguido por el nuevo nombre del archivo.

```
alulab@minix ~ $ ls
Descargas  Escritorio  milista      Música      Público
Documentos Imágenes   milista.old  Plantillas  Vídeos
alulab@minix ~ $ mv milista.old mi_lista.old
alulab@minix ~ $ ls
Descargas  Escritorio  milista      Música      Público
Documentos Imágenes   mi_lista.old Plantillas  Vídeos
alulab@minix ~ $ mv mi_lista.old milista.old
alulab@minix ~ $ ls
Descargas  Escritorio  milista      Música      Público
Documentos Imágenes   milista.old  Plantillas  Vídeos
alulab@minix ~ $
```

En el ejemplo anterior el comando *mv* se usó para cambiar el nombre de un archivo. Sin embargo, cuando los dos argumentos para este comando hacen referencia a directorios diferentes, entonces el archivo es movido del primer directorio al segundo.

Ejercicios

- 1.- Cambie la extensión a todos los archivos que fueron copiados en el apartado anterior (con extensión *.h*), por el de *.h.bak*
- 2.- Mueva los archivos que ha renombrado en el paso anterior, al directorio *Descargas*

Removiendo un archivo. El comando *rm*

Para eliminar o remover un archivo del sistema, emplee el comando *rm*. El argumento para *rm* es simplemente el nombre del archivo a ser removido.

```
alulab@minix ~ $ ls
Descargas  Escritorio  milista      Música      Público
Documentos Imágenes   milista.old  Plantillas  Vídeos
alulab@minix ~ $ rm milista.old
alulab@minix ~ $ ls
Descargas  Escritorio  milista  Plantillas  Vídeos
Documentos Imágenes   Música   Público
alulab@minix ~ $
```

Ejercicios

- 1.- Borre de forma imperativa (con un solo comando) todos los archivos que inicien con la letra *t* y que se encuentran en el directorio *Descargas* de su *home*.
- 2.- Ejecute el comando *rm* de forma que le pregunte uno a uno qué archivo desea borrar. Borre de forma alternada los archivos, cuando el *shell* le pregunte.
- 3.- En el directorio *Descargas*, borre el directorio *etc* y todo lo que contiene.

Cambiando los permisos a un archivo. El comando *chmod*

A diferencia de MS-Windows el hecho de que un archivo tenga una extensión *.com* o *.exe* no significa que sea un programa ejecutable. La acción de ejecutar cualquier programa esta condicionada al permiso correspondiente *x* de ejecución del archivo. Esto es importante a la hora de escribir programas, como los *scripts*. Un *script* es semejante a un archivo por lotes (*bat*) del DOS. Estos archivos son de tipo texto, y para que se ejecuten se le deberá colocar el permiso de ejecución con el comando *chmod*.

El comando *chmod*, cambia los permisos de acceso sobre archivos, sólo el propietario puede modificarlos. El formato es el siguiente:

```
chmod a|u|g|o +|- r|w|x <nombre de archivo>
```

La letra *a* (*all*) indica que el permiso se aplicará a todos: propietario, grupos *u* otros. Si se desea modificar el permiso solo al propietario se tendrá que escribir *u* (*user*) en lugar de *a*. En los demás casos emplee la letra *g* para grupo y *o* para otros. El signo *-* indica que se le está quitando los permisos, si desea añadirle en lugar de quitarle, escriba *+* en lugar de *-*. Y por último el siguiente carácter deberá ser: *r*, *w* ó *x* que indican el tipo de permiso que deseamos eliminar o agregar. A continuación primero quitaremos el permiso de escritura al archivo *milista* para todos (*all*) los usuarios, para luego restituirla:

```
alulab@minix ~ $ ls -l milista
-rw-rw-r-- 1 alulab alulab 15347 mar 14 20:32 milista
alulab@minix ~ $ chmod a-w milista
alulab@minix ~ $ ls -l milista
-r--r--r-- 1 alulab alulab 15347 mar 14 20:32 milista
alulab@minix ~ $ ls -l >> milista
-bash: milista: Permission denied
alulab@minix ~ $ chmod a+w milista
alulab@minix ~ $ ls -l milista
-rw-rw-rw- 1 alulab alulab 15347 mar 14 20:32 milista
alulab@minix ~ $ ls -l >> milista
alulab@minix ~ $ ls -l milista
-rw-rw-rw- 1 alulab alulab 15856 mar 14 20:33 milista
alulab@minix ~ $
```

Ejercicios

- 1.- Cambie los permisos de todos los archivos contenidos en Descargas de forma que cualquier usuario pueda modificarlos.
2. Cree un directorio con nombre *foo*, elimine el permiso de escritura para todos los casos al directorio *foo*. Ahora copie el archivo */etc/passwd* al directorio *foo*. ¿Qué sucede? Asigne el permiso de escritura para todos los usuarios, vuelva a copiar el archivo. ¿Qué sucede?
- 3.- Al directorio *foo* creando en el ejercicio anterior elimine el permiso de ejecución. Ingrese a dicho directorio. ¿Qué sucede?
- 4.- Con la ayuda de Intrnet averigüe sobre el permiso SUID. ¿Qué archivos tienen este permiso en el directorio */usr/bin*?
- 5.- Con la ayuda de Intrnet averigüe sobre el permiso SGID. ¿Qué archivos tienen este permiso en el directorio */usr/bin*?
- 6.- Con la ayuda de Intrnet averigüe sobre el permiso denominado *sticky bit*. ¿Qué directorios tienen este permiso en el disco duro?

Creando enlaces entre archivos. El comando *ln*

```
alulab@minix ~ $ ls -l milista
-rw-rw-rw- 1 alulab alulab 15856 mar 14 20:33 milista
alulab@minix ~ $ ln milista mi_lista
alulab@minix ~ $ ls -li milista mi_lista
2102636 -rw-rw-rw- 2 alulab alulab 15856 mar 14 20:33 milista
2102636 -rw-rw-rw- 2 alulab alulab 15856 mar 14 20:33 mi_lista
alulab@minix ~ $
```

Observe que tiene el mismo número de *inodo*

Se ha creado un *hard link* entre *milista* y *mi_lista*. En este caso se crea una nueva entrada con el nombre *mi_lista* en el directorio de trabajo, pero el número de *inodo* es el mismo que el de *milista*. Un *inodo* es una estructura en el que se guarda información acerca del archivo (excepto su nombre), datos como los permisos, tipo de archivo, fecha de creación, fecha de modificación, tamaño, etc. Son guardados en el *inodo*. Todo archivo tiene un *inodo*. El *inodo* es manejado por el sistema operativo y no es accesible al usuario de forma directa. Se puede decir que un enlace es un nuevo nombre para el mismo archivo. Normalmente se emplea para que diferentes usuarios tengan acceso al mismo archivo, sin tener que duplicar la información.

Existen otro tipo de enlaces llamados *soft-link*. A diferencia del primero este enlaza dos archivos no mediante su *inodo*, sino a través de un archivo que contiene la ruta y el nombre del archivo enlazado.

```
alulab@minix ~ $ ln -s /etc/passwd
alulab@minix ~ $ ls -l
total 64
drwxr-xr-x 2 alulab alulab 4096 mar 14 20:15 Descargas
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Documentos
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Escritorio
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Imágenes
-rw-rw-rw- 2 alulab alulab 15856 mar 14 20:33 milista
-rw-rw-rw- 2 alulab alulab 15856 mar 14 20:33 mi_lista
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Música
lrwxrwxrwx 1 alulab alulab 11 mar 15 16:30 passwd -> /etc/passwd
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Plantillas
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Público
drwxr-xr-x 2 alulab alulab 4096 mar 13 10:36 Vídeos
alulab@minix ~ $
```

El archivo *passwd* es un enlace al archivo */etc/passwd*. En este caso *passwd* y */etc/passwd* son dos archivos completamente distintos. El primero contiene la ruta donde se encuentra el segundo, sin embargo cada vez que usted accede al archivo *passwd* el sistema se comporta como si hubiera accedido al archivo */etc/passwd*.

Comprimiendo archivos. El comando *zip*

```
alulab@minix ~ $ ls
Descargas Documentos Escritorio Imágenes milista mi_lista Música passwd Plantillas Público Vídeos
alulab@minix ~ $ cp /usr/bin/haddock .
alulab@minix ~ $ ls
Descargas Escritorio Imágenes mi_lista passwd Público
Documentos haddock milista Música Plantillas Vídeos
alulab@minix ~ $ zip haddock.zip haddock
  adding: haddock (deflated 81%)
alulab@minix ~ $ ls
Descargas Escritorio haddock.zip milista Música Plantillas Vídeos
Documentos haddock Imágenes mi_lista passwd Público
alulab@minix ~ $ ls -l haddock*
-rwxr-xr-x 1 alulab alulab 35446584 mar 15 16:36 haddock
-rw-rw-r-- 1 alulab alulab 6773131 mar 15 16:36 haddock.zip
alulab@minix ~ $
```

Archivos de configuración.

Existe algunos archivos que permiten configurar la forma de trabajar en la terminal. El nombre de estos archivos normalmente inician con un punto ('.'). El sistema por defecto no los muestra, ni con el navegador de archivos (en forma gráfica) ni en la terminal (con el comando `ls`). Para poder verlos se deben de listar con la opción `-a`, o en el navegador presionando `Ctrl+h`.

```
alulab@minix ~ $ ls -a
.          .config    Escritorio  haddock    .local      passwd      .pulse-cookie
..         .dbus      Descargas   haddock.zip milista     Plantillas  Videos
.bash_history .bash_logout .dmrc       .ICEauthority mi_lista    .profile    .xsession-errors
.cache      Documentos .gtk-bookmarks .linuxmint  Música      .pulse
alulab@minix ~ $
```

Observe que son muchos archivos o directorios que inician con punto. Esto se debe por que muchos programas emplean archivos de configuración y con el uso de nuevos programas, estos archivos se incrementan.

PROCESOS

Un proceso es un programa en ejecución. Cada proceso posee un número identificador llamado *pid*.

1. El comando `ps` muestra en pantalla el estado de los procesos activos. Ejecute el comando

```
ps ax
```

obtendrá una salida semejante a la siguiente:

PID	TTY	STAT	TIME	COMMAND
1483	tty7	Ss+	80:09	/usr/bin/Xorg :0 -br
2107	tty1	Ss+	0:00	/sbin/getty 38400 tty1
2110	tty4	Ss+	0:00	/sbin/getty 38400 tty4
2111	tty5	Ss+	0:00	/sbin/getty 38400 tty5
2112	tty6	Ss+	0:00	/sbin/getty 38400 tty6
30387	pts/0	Ss	0:00	bash
31058	pts/0	S+	0:00	man ps
31067	pts/0	S+	0:00	pager -s
31159	pts/1	Ss	0:00	bash
31250	pts/1	R+	0:00	ps a

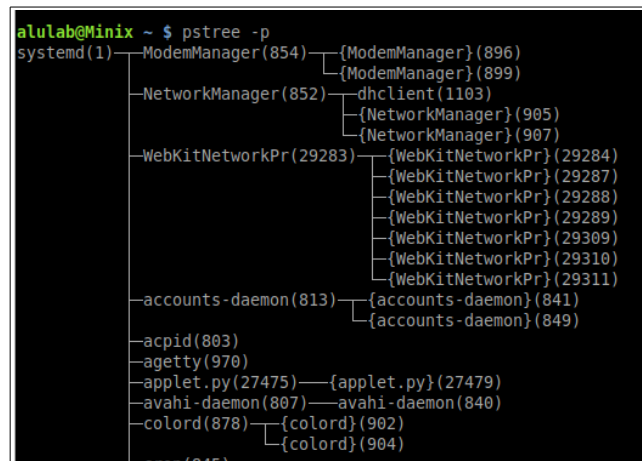
La primera columna indica el identificador del proceso (*pid*), la segunda columna especifica la terminal o pseudo terminal donde se ha ejecutado el comando. La tercera columna indica el estado del programa, la cuarta columna indica el tiempo que se ha ejecutado y la última el nombre del programa.

2. Haciendo uso del manual en línea, encuentre la forma de mostrar sólo las siguientes columnas:

```
PID    PPID   TTY     STAT   TIME   COMMAND
```

3. También puede obtener por pantalla el árbol de procesos. Escriba en la terminal:

```
ps tree -p
```



La salida mostrada está recortada, pues el árbol de procesos es muy grande. Observe el identificador (se encuentra entre paréntesis) del proceso (*pid*) que tiene por nombre *systemd*. Esto indica que es el proceso “padre” de todos los procesos de usuario.

4. El comando `kill` envía una señal a un proceso. El proceso puede ignorar la señal, atrapar la señal y luego hacer una tarea previamente programada o el proceso simplemente termina. La acción por defecto es la de terminar. Para identificar el proceso se usa su PID. La señal 9 y 15 ocasionan que el proceso termine. A veces ésta es una forma de eliminar un proceso que se ha quedado bloqueado o que no responde. En una terminal (Ctrl + Alt + t). Por ejemplo, ejecute el programa *xed*:

```
$ xed &
```

En otra terminal (Ctrl + Alt + t) ejecute el comando `ps ax`. Identifique cuál es el PID del proceso *xed*. Luego elimine el proceso *xed*, suponga que el *pid* de *xed* es 298, entonces escriba en la terminal:

```
$ kill -9 298
```

5. A veces es engorroso buscar el *pid* deseado en una lista muy larga como la que emite `ps ax`. En ese caso puede usar:

6. Ahora con el *pid* obtenido ejecute `pstree` tal como se muestra a continuación:

7. Usted puede ver de forma dinámica la ejecución de los procesos. En una terminal escriba: `top`

```
top - 18:51:41 up 4 days, 2:58, 2 users, load average: 0,02, 0,08, 0,13
Tareas: 270 total, 1 ejecutar, 203 hibernar, 0 detener, 0 zombie
%Cpu(s): 3,3 usuario, 1,3 sist, 0,0 adecuado, 95,3 inact, 0,0 en espera, 0,0 hardw int, 0
KiB Mem : 8068380 total, 2555720 libre, 1759024 usado, 3753636 búfer/caché
KiB Intercambio: 20115448 total, 20115448 libre, 0 usado, 5769892 dispon Mem
```

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN
7107	alejand+	20	0	3720712	150012	70212	S	19,9	1,9	28:38.35	cinnamon
6746	root	20	0	650572	101564	68868	S	3,7	1,3	7:23.84	Xorg
29450	alejand+	20	0	1068376	178968	41584	S	0,7	2,2	0:12.80	shutter
31895	alulab	20	0	46864	4276	3544	R	0,7	0,1	0:00.11	top
7434	alejand+	20	0	716392	75928	38236	S	0,3	0,9	0:43.27	terminator
9811	alejand+	20	0	1822940	200736	123728	S	0,3	2,5	3:28.60	Web Content
9856	alejand+	20	0	1593424	138792	87140	S	0,3	1,7	2:05.34	WebExtensions
24508	root	20	0	0	0	0	I	0,3	0,0	0:00.58	kworker/1:1
31842	root	20	0	0	0	0	I	0,3	0,0	0:00.03	kworker/u16:0
1	root	20	0	225776	9404	6536	S	0,0	0,1	0:05.30	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.15	kthreadd
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq
7	root	20	0	0	0	0	S	0,0	0,0	0:00.13	ksoftirqd/0
8	root	20	0	0	0	0	I	0,0	0,0	0:48.10	rcu_sched
9	root	20	0	0	0	0	I	0,0	0,0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0,0	0,0	0:00.00	migration/0

8. Una lista de comandos puede ejecutarse en una sola línea si se les separa por punto y coma. Por ejemplo si se desea aclarar pantalla y luego de 2 segundos hacer un listado de los archivos del directorio de trabajo, escriba:

```
$ clr; sleep 2; ls -l
```

o puede agrupar con paréntesis de acuerdo a su conveniencia. Por ejemplo:

```
$ ls -a|(clr; wc -l) > data;echo archivos en directorio actual >> data; cat data
```

En algunas ocasiones el comando es muy extenso y se desea escribir en dos líneas. En estos casos se coloca el *backslash* (también conocido como *slash* invertido) al final de la primera línea y a continuación se presiona la tecla <Enter> luego se continúa en la segunda línea. El objetivo del *backslash* es evitar que el *shell* interprete el carácter de cambio de línea (generado al presionar <Enter>) que hay a continuación.

Al inicio de la segunda línea, después de presionar la tecla <Enter>, no aparece el **prompt primario** (\$), sino el **prompt secundario** (>).

9. El *shell* tiene muchos comando y cuenta con diferentes herramientas que nos permiten llevar a cabo tareas complejas de forma simple, al estilo UNIX. Por ejemplo, si usted necesita saber los nombres de los archivos fuentes escritos en C en todo el sistema de archivos, que contienen la llamada al sistema *fork*,:

```
alulab@Minix ~ $ find / -name '*.c' -exec grep -l fork {} \; 2>/dev/null
/usr/include/X11/Xtrans/Xtranslcl.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-native/adainclude/terminals.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-native/adainclude/expect.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-native/adainclude/adaint.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-sjlj/adainclude/terminals.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-sjlj/adainclude/expect.c
/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/rts-sjlj/adainclude/adaint.c
```

10. Un proceso se encuentra ejecutándose en **primer plano** (*foreground*) si interactúa con el usuario recibiendo las entradas del teclado y enviando las salidas a la pantalla (mientras no se redirijan). En cambio un proceso se ejecuta en **segundo plano** (*background*) si no hace uso de la consola, dejándola libre para ejecutar algún otro comando. Lanzar un proceso en *background* es útil cuando éste puede tomar mucho tiempo en ejecutarse. En símbolo "&" al final de un comando indica que estos se ejecutarán en segundo plano.

En el siguiente ejemplo la ejecución se llevará a cabo en *background*, esto se notará inmediatamente porque el *prompt* quedará libre y a continuación se podrá ejecutar el comando *ps*, para observar la ejecución en *background*. Por último, puede ver la salida, que a propósito se ha enviado a otra terminal virtual.

Observación: Para poder ejecutar este comando, primero debe abrir dos terminales virtuales y con la ayuda del comando *ps ax*, averiguar cual es la seudo terminal que está empleando. Para el ejemplo una terminal correspondía a */dev/pts/4* y la otra a */dev/pts/6*. En la terminal identificada como */dev/pts/4* se escribió el siguiente comando para que la salida se muestre en la terminal identificada como */dev/pts/6*. Usted debe identificar cuáles son las seudo terminales que está empleando.

```
$ (find / -name '*.ch' 2>/dev/null | \
> while read file;do cat $file > /dev/pts/6 2>/dev/null;done) &
```

Aquí se presiona <Enter> e inmediatamente se escribe en la siguiente línea

Este es el prompt secundario, usted no debe escribirlo.

```
$ ps
```


¿A qué corresponde la salida en la segunda terminal?

11. A veces es muy útil filtrar los resultados mostrados por pantalla. Por ejemplo si está buscando el nombre de alguna aplicación que se encuentra en `/usr/bin` y ejecuta el comando `ls -l`, es posible que la lista mostrada sea lo bastante larga como para tornar la búsqueda en lenta y pesada. Sin embargo si usted se acuerda parte del nombre podría filtrar la salida. Por ejemplo:


```
$ ls -l /usr/bin | grep grub
```

Mostrará por pantalla todos los nombres de archivos que contengan la cadena `grub`. El comando `grep` no solo acepta cadenas sino expresiones regulares. Es más según los autores el comando `grep` fue escrito en una noche al modificar el comando `ed`. Por ejemplo:

```
$ ls -l /usr/bin | grep '^l'
```

Mostrará todas las salidas que inician con la letra 'l'

Investigue acerca del comando `cut` (use el manual en línea o busque en Internet) y con ayuda de este muestre por pantalla solo el modo de acceso y el nombre de cada archivo que coincidan con el siguiente patrón: “nombres de archivo que inicien con la letra *a*”.

Importante: cierre todas las terminales virtuales .

12. Copie el archivo `.bash_history` con el de nombre `historial_sesion`. Comprima dicho archivo con formato `zip`. Esto lo puede hacer desde el *Navegador de archivos*, haciendo *click* derecho sobre el archivo correspondiente y eligiendo la opción **Crear archivador**. Coloque por nombre, su código de alumno, por ejemplo: `20000543.zip` y añada el archivo a Intranet en la carpeta **Buzón** que se encuentra en la sección de **Documentos de INF239-Sistemas Operativos**.

Prof. Alejandro T. Bello Ruiz