UNIVERSIDAD NACIONAL DE HURLINGHAM

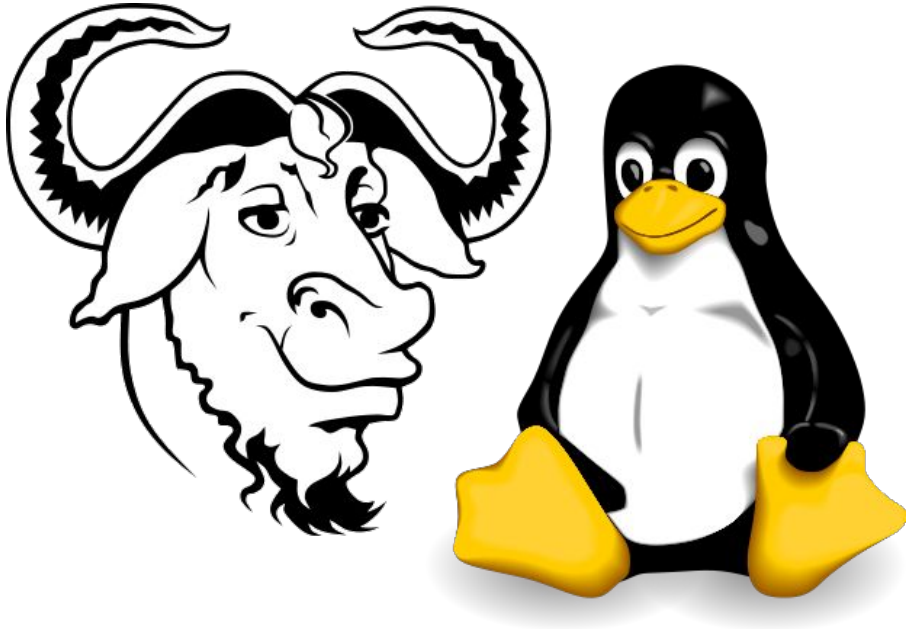
Universidad Nacional de San Martín

Linux environment and vi editor

Leandro Luciano Gagliardi
lgagliardi@unsam.edu.ar

Conociendo GNU/Linux

- GNU/Linux es un sistema operativo derivado de UNIX, que se distribuye en forma libre.



¿Qué es UNIX?

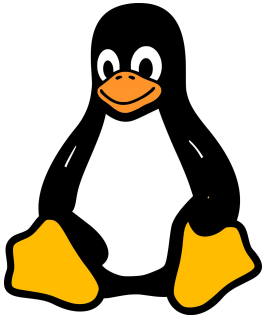
UNIX es un sistema operativo multitarea, multiusuario, creado en 1969 por los investigadores Thompson y Ritchie de los Laboratorios Bell, en los Estados Unidos. Las primeras versiones fueron escritas en **assembler**, pero muy pronto fue re-escrito en **lenguaje C**.
En sus primeros años, no se lo utilizó comercialmente, sino que se lo usaba para proyectos de investigación en laboratorios y se distribuía gratuitamente en las universidades, donde tuvo mucha aceptación.
En 1975, Bell decidió comercializarlo. Dado que el sistema se vendía con una licencia que permitía modificarlo y redistribuirlo, a lo largo del tiempo fueron surgiendo una gran variedad de sistemas derivados del UNIX original.
Por esta razón, varias veces se hizo necesario normalizar estos sistemas, para que cumplan con determinadas normas (POSIX, UNIX95, etc), para permitir la compatibilidad entre los diferentes sistemas.
De estas normas, el sistema operativo **GNU/Linux** satisface la norma **POSIX-1**, y casi completamente la **POSIX-2**.

¿Qué es GNU?

La sigla GNU significa GNU is Not Unix.
En 1984, Richard Stallman fundó el Proyecto **GNU** con el objetivo de conseguir un sistema operativo libre y abierto. Esto es, un sistema operativo tal que los usuarios puedan usarlo, leer el código fuente, modificarlo, y redistribuirlo.
A partir de ese momento, un gran número de colaboradores se fueron sumando al proyecto, desarrollando software libre para reemplazar cada una de las herramientas del sistema UNIX.
La filosofía GNU apoya el crecimiento de la sociedad como un conjunto, haciendo especial hincapié en la valoración de las libertades personales, aún cuando esto puede estar en conflicto con intereses empresariales.

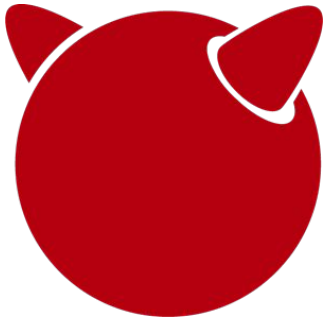
¿Qué es Linux?

En 1991, Linus Torvalds completó el sistema con su kernel (que es la aplicación encargada de comunicar los procesos con el hardware de la computadora). A este kernel lo bautizó Linux. De esta manera, se formó el sistema **GNU/Linux**.



¿Qué es BSD?

La Universidad de Berkeley estuvo relacionada con el desarrollo de los sistemas operativos UNIX.
Recibió de AT&T una versión gratuita de UNIX, y a partir de entonces comenzó a promover el desarrollo de aplicaciones para UNIX dentro de la universidad.
Más adelante, desarrolló su propio sistema operativo UNIX, sin utilizar el código fuente de AT&T.
El kernel fue creado desde Berkeley, pero las herramientas utilizadas son en su mayoría GNU, es decir las mismas que en el sistema GNU/Linux.
Existen actualmente 3 sistemas operativos libres, derivado de BSD: FreeBSD, OpenBSD y NetBSD.



¿Qué es X?

El sistema operativo GNU/Linux cuenta con una interfaz gráfica, llamada XFree86 o simplemente X. El protocolo X fue desarrollado por el MIT (**M**assachusetts **I**nstitute of **T**echnology), principalmente como un logro académico para proporcionar un entorno gráfico a UNIX. La licencia mediante la cual se distribuye permite usarlo, modificarlo, redistribuirlo e incluso re-licenciarlo.

¿Qué son las distribuciones?

El código fuente del sistema **GNU** y del kernel **Linux** está accesible a todo el mundo, sin embargo, hacer funcionar un sistema a partir del código fuente es bastante difícil. Por eso, un sistema operativo se distribuye (normalmente) en formato binario, es decir ya compilado.
Poco después de que apareciera el kernel de Linux, comenzaron a aparecer las primeras distribuciones, que agrupaban versiones probadas de varios programas, junto con el kernel, de tal manera que formaban un sistema operativo listo para usar.
A medida que fue pasando el tiempo, algunas distribuciones se fueron haciendo más sofisticadas, otras desaparecieron y otras se hicieron comerciales.
Cada usuario de GNU/Linux suele elegir la distribución con la que se siente más cómodo, y no tiene sentido entrar en discusiones acerca de cuál es mejor.

En particular, nosotros usaremos Ubuntu para trabajar.

Ejecuten el siguiente comando para verificar la versión de Ubuntu:

$ lsb_release -a

¿Qué son las plataformas?

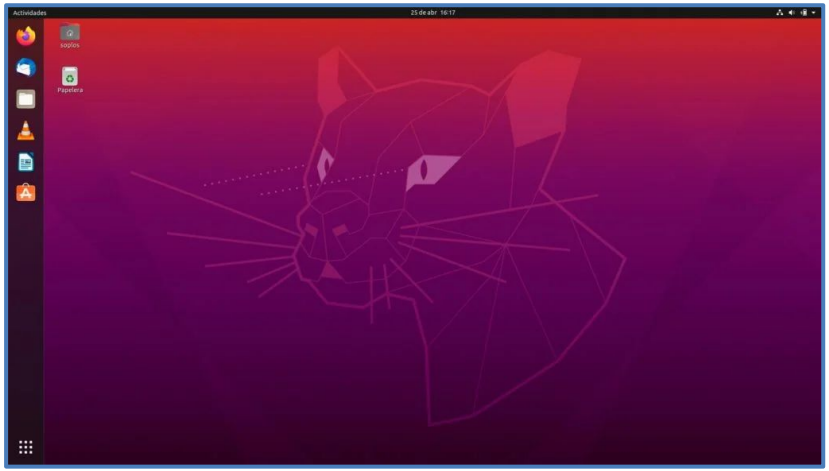
El mundo de las computadoras no se restringe a las Computadoras Personales con las que estamos en contacto todos los días. Existen diversas arquitecturas en las que una computadora se puede presentar. A cada una de estas arquitecturas la llamamos plataforma.
Ejemplos de algunas plataformas posibles son: iMac (de Macintosh), Sparc (de Sun), S/390 (de IBM), PlayStation (de Sony), XBox (de Microsoft).

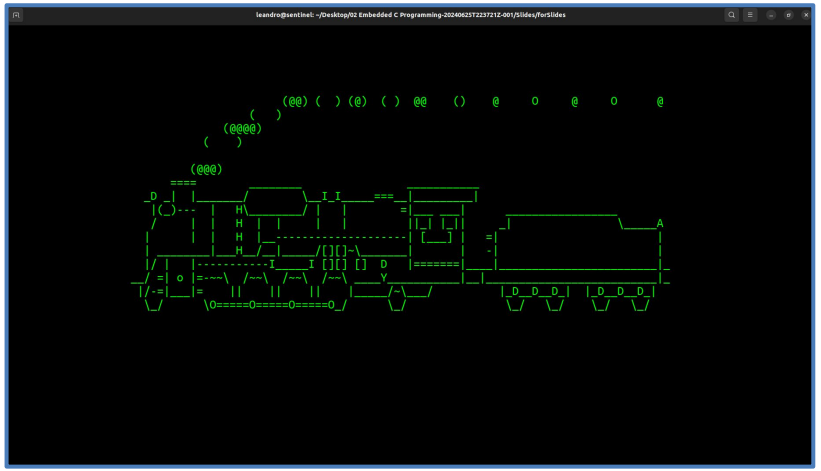
Ejecutar el siguiente comando para verificar la arquitectura del sistema:

$ uname -m

Modo gráfico - Modo consola

Como ya dijimos anteriormente, GNU/Linux puede utilizar el Modo Gráfico, si utiliza la aplicación XFree86.
Por otro lado, llamamos Modo Consola, al modo que es puramente texto. En el presente curso usaremos principalmente este modo.





Recorrido por el sistema

Vamos a ver algunos comandos básicos de todo UNIX, que nos permitirán familiarizarnos con el sistema.
La mayoría de estos comandos son herramientas simples, que realizan una sola tarea. Más adelante aprenderemos cómo combinar lo que hacen dos o más herramientas para lograr un resultado más interesante.
Algunos de estos comandos pueden recibir opciones o parámetros, que pueden hacerlos realizar tareas diferentes. En UNIX, casi todas las opciones que reciba un comando estarán precedidas por el carácter **-**, y pueden combinarse según sea necesario.
Es importante notar que UNIX es sensible a las mayúsculas y minúsculas (case sensitive), de forma que si queremos obtener la salida correcta es necesario escribir los comandos que aquí se explican tal cual se presentan (casi siempre en minúsculas).

Comandos Sencillos

- **$ date**
  Ejecutando el comando **date** obtendremos la fecha y la hora actual.
- **$ who**
  who nos permite ver los usuarios que están utilizando el sistema, incluyendo la fecha en la que se conectaron al sistema.
- **$ uptime**
  Podemos ver cuánto tiempo hace que se ha iniciado el sistema de nuestra computadora ejecutando el comando **uptime**. También obtendremos alguna información adicional, como la cantidad de usuarios que están utilizando el sistema, o la carga promedio que tiene el procesador.
- **$ clear**
  El comando **clear** sirve para limpiar la pantalla.
- **$ echo**
  echo es un comando muy sencillo. Lo que hace es repetir todo lo que recibe por línea de comandos. Si ejecutamos: **$ echo Hola** obtendremos la salida **Hola**.

Comandos relacionados con archivos y directorios

- **$ pwd**

  El comando **pwd** es muy sencillo, nos muestra la ruta de directorios en la que estamos situados en este momento. Por ejemplo, /home/user.

- **$ ls**

  Para listar los archivos dentro de un determinado directorio utilizaremos el comando **ls**. Si ejecutamos **ls** sin ninguna opción, nos listará los archivos del directorio actual. Si, en cambio, ejecutamos **ls directorio**, nos listará los archivos de ese directorio.
  Puede recibir varias opciones:
  - **-l**: muestra mucha más información sobre los archivos, como el usuario y el grupo al que pertenece, el tamaño, los permisos, etc.
  - **-a**: muestra todos los archivos, incluyendo los ocultos.
  - **-t**: ordena los archivos por fecha de modificación.

  Estas opciones pueden combinarse, utilizando un solo signo '-', por ejemplo: **$ ls -lat**.

Comandos relacionados con archivos y directorios

- **$ touch**

  El comando **touch archivo** puede tener dos consecuencias: si **archivo** no existe aún, lo crea con tamaño 0 y como propiedad de nuestro usuario. Por otro lado, si archivo ya existe, actualiza la fecha de modificación.

- **$ cp**

  El comando **cp** es el que se utiliza para copiar archivos. Si escribimos **$ cp viejo nuevo**, copiaremos el archivo **viejo** con el nombre **nuevo**. Es decir, el archivo origen se escribe primero y a continuación el archivo que se va a crear. Una vez hecha la copia, tendremos dos archivos diferentes, con el mismo contenido. Por otro lado, también podemos ejecutar **$ cp archivo1 archivo2 directorio**.

- **$ mv**

  Muy similar a **cp**, el comando **mv** es el que se utiliza para mover archivos de un lugar a otro, o para cambiarle el nombre a un archivo. Si ejecutamos, **$ mv viejo nuevo**, el archivo **viejo** habrá pasado a llamarse **nuevo**. Por otro lado, si ejecutamos **$ mv archivo1 archivo2 directorio**, los archivos **archivo1** y **archivo2** se moverán dentro de **directorio**.

Comandos relacionados con archivos y directorios

- **$ rm**

  Para borrar archivos utilizamos el comando **rm**. Hay que usarlo cuidadosamente, porque una vez que los archivos han sido borrados, no pueden recuperarse de ninguna forma. Si deseamos que **rm** nos pregunte si queremos borrar o no un archivo, debemos utilizar la opción **-i**, mientras que si deseamos que no nos pregunte utilizamos la opción **-f**. Dependerá de la configuración del sistema cuál de estas dos opciones es la que está seleccionada por omisión. Para borrar un directorio ejecutamos **$ rm -r directorio**.

- **$ mkdir**

  Utilizamos el comando **mkdir directorio** para crear directorios. Pueden utilizarse rutas absolutas o relativas. Es decir que si queremos crear el directorio **/home/user/temp**, y estamos situados dentro del directorio /home/user, podremos ejecutar **mkdir temp** o **mkdir /home/user/temp** indistintamente.

- **$ rmdir**

  Para borrar directorios utilizamos el comando **rmdir directorio**. Solamente funcionará si el directorio está vacío. En caso contrario, habrá que borrar primero los archivos, para luego borrar el directorio.

Comandos relacionados con archivos y directorios

- **$ cd**

  El comando **cd** nos permite cambiar de directorio.
  Por ejemplo, **$ cd /** nos lleva al **root**, que es de donde surgen todos los directorios del sistema.
  Para cambiar a otro directorio dentro del árbol, podemos utilizar **cd usr**, o **$ cd /home/user**. Más adelante veremos cómo se organiza el árbol de directorios, y qué hay en cada uno.
  Utilizado sin ningún otro parámetro, **$ cd** nos lleva al directorio personal del usuario (**home**). Otra manera de ir al directorio personal es utilizar **$ cd ~**, ya que el símbolo **~** identifica al directorio de cada usuario.
  Para cambiar al directorio padre del directorio actual se utiliza **$ cd ..** (no olvidar el espacio).
  Mientras que para volver al directorio en el que nos encontrábamos antes de ejecutar el último **cd**, podemos ejecutar **$ cd -**.

Comandos relacionados con archivos y directorios

- **$ file**

  En UNIX los archivos no se identifican por su extensión. Se les puede poner una extensión, pero es de adorno.
  El comando **file** nos permite tener una idea del contenido de un archivo. Al ejecutar **$ file archivo**, inspecciona partes del archivo para darse cuenta qué clase de archivo es.
- **$ du**

  El comando **du**, Disk Usage, nos muestra el espacio que ocupan todos los directorios a partir del directorio actual. El número de la primera columna es el espacio ocupado por el directorio y está expresado en kB.
  - **-s** nos muestra únicamente el total.
  - **-a** muestra lo que ocupan los archivos, además de los directorios.
  - **-h** hace el listado, indicando la unidad (human readable).
  - **$ du archivo** nos dice cuánto ocupa el archivo.
- **$ df**

  El comando **df**, Disk Free, nos muestra qué tan ocupados están los discos del sistema.

Comandos relacionados con archivos y directorios

- **$ find**

  El comando **find** permite encontrar archivos, utilizando diversas técnicas. En principio, si se le pasa como parámetro únicamente una determinada ruta, por ejemplo **$ find /home/user**, el comando buscará todos los archivos y directorios que se encuentren a partir de esa ruta.
  Utilizando algunos otros parámetros es posible buscar los archivos por diversos criterios. Ejemplos:

  - **$ find . -name "hola.txt"** encuentra todos los archivos llamados hola.txt que se encuentren a partir del directorio actual.
  - **$ find . -size 50k** busca los archivos que ocupan 50 kilobytes a partir del directorio actual. Si se utiliza **$ find . -size 20c**, buscará los archivos que ocupen 20 bytes. Y si se utiliza **$ find . -size 5b**, buscará los archivos que ocupen 5 bloques de 512 bytes cada uno.
  - **$ find /home/user -empty** busca todos los archivos que se encuentran vacíos, a partir del directorio **/home/user**.

Comandos relacionados con archivos y directorios

- **$ locate**

  Las búsquedas de **locate** nos sirven para encontrar archivos en cualquier parte del sistema, por ejemplo **$ locate bash** nos va a mostrar todos los archivos y directorios del sistema que en su nombre contienen la palabra **bash**.
  Las búsquedas de **locate** son mucho más rápidas que las de **find**, esto se debe a que utiliza una base de datos que contiene los nombres de los archivos de todo el sistema. Esta base de datos no se actualiza constantemente, por lo que locate no encontrará los archivos recién agregados al sistema e incluirá los recientemente removidos hasta que esta base se actualice.
  Para actualizar esta base de datos debemos ejecutar **$ updatedb**, como superusuario. En las máquinas que están prendidas constantemente, este programa se ejecuta una vez al día, durante la madrugada, de manera que la base de datos permanece actualizada.

Comandos relacionados con archivos y directorios

- **$ od**

  El comando od Octal Dump, nos permite ver byte a byte el contenido de un archivo.
  - **$ od archivo** nos muestra el contenido del archivo expresado en números octales, generalmente tomados de a dos bytes.
  - **$ od -b archivo** nos muestra el contenido, en números octales, byte a byte.
  - **$ od -c archivo** nos muestra los caracteres que forman el archivo, uno por uno.
  - **$ od -cb archivo** nos muestra los caracteres, y debajo de cada carácter el número octal del byte.
  - **$ od -h archivo** nos muestra el contenido, en números hexadecimales, tomados de a dos bytes.
- **$ wc**

  El comando **$ wc archivo**, se utiliza para contar la cantidad de líneas, palabras y letras que tiene un **archivo**.

Comandos relacionados con archivos y directorios

- **$ less**

  El comando **less** permite paginar la salida de otros comandos, o bien, el contenido de algún archivo.
  Si ejecutamos **$ less archivo**, veremos la primera página del archivo. Si este archivo es lo suficientemente largo, podremos movernos hacia abajo y hacia arriba. con **q** salimos de la visualización.

Comandos relacionados con procesos

Cada aplicación que se ejecuta dentro de un sistema UNIX es un proceso. Algunos procesos están ejecutándose sin que nosotros lo sepamos. Otros procesos comienzan a ejecutarse cuando nosotros se lo indicamos.
Cada proceso que se ejecuta dentro de nuestra computadora tiene un número que lo identifica, llamado Process ID (PID). Este número será el que deberemos utilizar para referirnos a ese proceso con otros comandos.

- **$ top**

  El comando **top** nos muestra algunos de los procesos que se están ejecutando, ordenados por el tiempo de procesador de la computadora que consumen. Muestra algunos datos adicionales de cada proceso, por ejemplo, en la primera columna, podemos observar el PID de cada uno. Además, muestra otros datos acerca del uso que se le está dando a la máquina.
  Para salir: **q**.

Comandos relacionados con procesos

- **$ ps**

  El comando **ps** nos muestra la lista de procesos que se están ejecutando en la computadora. En particular, es interesante ver la salida de **$ ps ax**, que nos muestra todos los procesos, tanto los de nuestro usuario como los de otros, e incluso los procesos que no tienen usuario.
  La primera columna de la salida de ps también contiene el PID del proceso.

- **Ctrl-z**

  La combinación de teclas **Ctrl-z** sirve para suspender una tarea dentro de su ejecución. Por ejemplo, si estamos ejecutando el proceso **top** y presionamos **Ctrl-z**, se suspenderá la ejecución de **top** y volveremos a obtener la línea de comandos. Antes de devolvernos la línea de comandos, nos indicará cuál es el número de trabajo del proceso que hemos suspendido.

Comandos relacionados con procesos

- **bg / fg**

  El comando **bg** permite que el proceso que se halle suspendido, continúe ejecutándose en background (de fondo). Mientras que el comando **fg** permite que un proceso suspendido pase a foreground (a la pantalla principal).

- **$ jobs**

  Para poder ver qué comandos se están ejecutando en background y cuáles han sido suspendidos (o detenidos), podemos utilizar el comando **jobs**. La lista que nos muestra este comando incluye el número de trabajo, que no es el mismo que el PID, y podemos utilizar este identificador para decidir cuál de las tareas pasar a foreground o background. Por ejemplo: **$ fg 1** pasa a foreground el trabajo número 1. Mientras que **$ bg 3** pasa a background el trabajo número 3.

- **kill**

  Es para interrumpir la ejecución de un determinado proceso. El comando envía una señal al proceso por la cual se cierra. Podemos ejecutarlo teniendo en cuenta el PID del proceso. Por ejemplo: **$ kill 1234**, mataría el proceso 1234. O bien, teniendo en cuenta el número de trabajo. En ese caso: **$ kill %2**, detendrá el trabajo número 2.

Comandos relacionados con procesos

- **Ctrl-c**

  Cuando una aplicación se está ejecutando en foreground, y deseamos detenerla, podemos utilizar la combinación de teclas **Ctrl-c**.
  El uso de esta combinación es equivalente a ejecutar el comando **kill** con el número de proceso de la aplicación que estamos utilizando.

Obteniendo más información

- **$ man**

  Un comando muy importante es **man**. Este comando nos mostrará las hojas del manual del programa que estamos queriendo buscar. Por ejemplo, **$ man date** nos mostrará el manual del comando **date**, que ya sabemos que sirve para ver y configurar la fecha, aquí está explicado cómo utilizarlo.

- **$ info**

  Un comando muy similar a **man**, es el comando **info**. Las páginas que nos muestra este comando suelen tener una mayor cantidad de información acerca de la aplicación sobre la cual estamos consultando.

- **$ help**

  Algunos comandos son parte interna del intérprete de comandos, y por esta razón no tienen una página del manual que los explique.

  Para saber de qué manera utilizar estos comandos, usamos **help**. La ayuda que nos da este comando es más sintética que la de **man**. Por ejemplo **$ help jobs**, nos informará sobre el uso del comando **jobs** visto anteriormente.

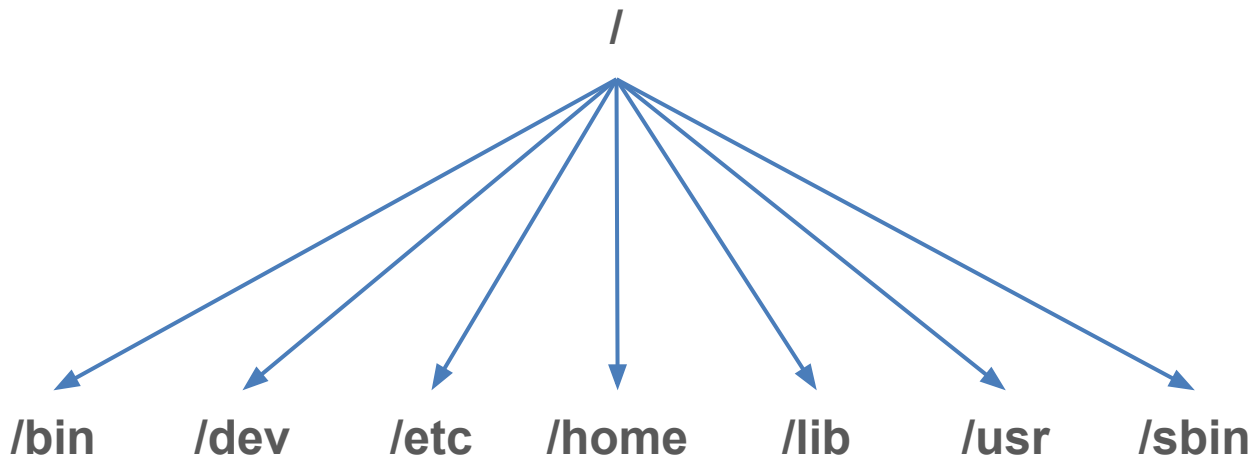
Obteniendo más información

- **Archivos con información**

  Dentro del directorio **/usr/share/doc**, encontramos una gran cantidad de documentos que tratan las distintas aplicaciones que tenemos instaladas en nuestro sistema. Ejecutando **$ cd /usr/share/doc** podemos dirigirnos al directorio en cuestión.

Árbol de Directorios

Existen diferentes maneras de ordenar la información dentro de los directorios. Veremos un orden básico que la mayoría de las distribuciones utilizan.



Árbol de Directorios

- **/bin** contiene los archivos ejecutables básicos del sistema. En este directorio, por ejemplo, encontramos algunos de los comandos de shell que nombramos.
- **/dev** contiene los archivos que representan a dispositivos. Los archivos que se encuentran en este directorio están relacionados con periféricos de la máquina, por ejemplo: **/dev/tty** representa la consola actual que se está usando. Así: **$ echo hola > /dev/tty** nos mostrará la misma salida que un simple **$ echo hola**.
- **/etc** contiene los archivos de configuración del sistema y de muchos de los programas instalados en el sistema. Además, contiene los scripts de inicio que se ejecutan cuando arranca la máquina. Generalmente los archivos que se encuentran en este directorio pueden ser editados sólo por el administrador de la máquina, es decir el superusuario, normalmente llamado “root”.
- **/home** en este directorio se encuentran los directorios principales de los usuarios. En estos directorios los usuarios tienen permisos de leer, escribir y ejecutar según deseen.

Árbol de Directorios

- **/lib** contiene las librerías necesarias para ejecutar los comandos que se encuentran en **/bin** y **/sbin**. Las librerías son rutinas que los programas utilizan frecuentemente, y pueden ser compartidas por varios programas al mismo tiempo.
- **/usr** contiene archivos que serán utilizados una vez que el sistema ya está funcionando. Por dentro se subdivide nuevamente en un árbol muy parecido al del directorio raíz. Encontramos, además, directorios como **src** (donde se suelen guardar los archivos con el código fuente del sistema), **games** (que tiene juegos), **share** (que contiene archivos a los que pueden acceder todos los programas y todos los usuarios).
- **/sbin** contiene los archivos ejecutables que son necesarios para poder administrar el sistema.

Un comando esencial: chmod

Para cambiar los permisos de un archivo utilizamos el comando **chmod**. Solamente el dueño de un archivo (o el administrador del sistema) puede cambiarle los permisos a un archivo. Aún si no tiene ninguno de los permisos explicados, el dueño del archivo siempre tiene permiso de cambiar los permisos de un archivo.

$ chmod nnn archivo

u g o

n | n | n = rwx | rwx | rwx = 111 | 101 | 001 = **751**

$ chmod 731 archivo

Esto significa que estamos otorgando al dueño del archivo permisos de escritura, lectura y ejecución; a grupos lectura y ejecución y a otros solo de ejecución.

pd: podemos verificar los permisos con **$ ls -l**

vi (o vim)

Vi es un editor de texto para consola. Es el editor de texto tradicional de UNIX, y en muchos sistemas es el único disponible, de manera que es importante saber usarlo.
Para comenzar a editar un archivo deberemos escribir: **$ vi archivo**, o bien ejecutar **$ vi**, y luego abrir el archivo con el comando adecuado.
En **vi** existen dos modos de trabajo: un modo de edición y un modo de comandos. Al iniciar el programa, estamos en el modo de comandos. Para ingresar al modo de edición debemos apretar **i**, o bien, **Insert**. Para volver al modo de comandos, utilizamos la tecla **ESC**.

Comandos básicos

- **:e archivo** abre el archivo.
- **:q** sale del programa, solo si ya se grabaron los cambios.
- **:q!** sale del programa sin grabar los cambios.
- **:w** graba el archivo.
- **:w archivo** graba el archivo con ese nombre (eq. Guardar Como)
- **:wq** graba el archivo y luego sale del programa.

Teclas de movimientos:

0: inicio de línea.
b: anterior palabra.
G: fin de archivo.

$: fin de línea.
w: próxima palabra.
gg: inicio de archivo.

Comandos básicos

Manejo de texto:

dd: corta la línea.
dw: corta la próxima palabra.
d$: corta hasta el final de la línea.
p: pega lo que se haya cortado o copiado
u (undo): deshace la última acción.
yy: copia la línea.
x: corta el caracter.

Uso de vim para un código en C

El sistema operativo **Unix/Linux** incluye un compilador de línea de comandos de **C**. Llamado **gcc**.

- Creación de programas (con el editor **vim**):

  $ vim filename.c

  El archivo se guarda presionando **ESC** y **SHIFT+zz** o bien **ESC**, **:** y **wq**, o bien **ESC**, **:** y **x**.
- Compilación de programas:

  $ gcc filename.c

  Si el programa tiene una función matemática, se compila como:

  $ gcc filename.c -lm

  Después de la compilación, el código ejecutable se almacena en el archivo **a.out**. Otra forma de compilación que proporciona un nombre de archivo ejecutable es:

  $ gcc filename.c -o executablename

  Después de la compilación, el código ejecutable se almacena en el archivo **executablename**.
- Ejecución de programas:

  $ ./a.out

  Si se compila con la opción **-o**, se puede ejecutar como:

  $ ./executablename

UNIVERSIDAD NACIONAL DE HURLINGHAM

Universidad Nacional de San Martín

Extra Slides

Leandro Gagliardi
lgagliardi@unsam.edu.ar

¿Qué es POSIX?

- Al trabajar con Linux, a menudo encontraremos referencias a POSIX.
- POSIX (**P**ortable **O**perating **S**ystem **I**nterface). Consiste en una familia de estándares especificadas por la IEEE con el objetivo de facilitar la interoperabilidad de sistemas operativos. Además, POSIX establece las reglas para la portabilidad de programas. Por ejemplo, cuando se desarrolla software que cumple con los estándares POSIX existe una gran probabilidad de que se podrá utilizar en sistemas operativos del tipo Unix. Si se ignoran tales reglas, es muy posible que el programa o librería funcione bien en un sistema dado pero que no lo haga en otro.
- Por ejemplo, en la especificación **IEEE 1003** (designación formal de los estándares POSIX), al buscar la referencia sobre el comando **du**, podemos ver que solamente las opciones **-a**, **-H**, **-k**, **-L**, **-s**, y **-x** son obligatorias. Otras, tales como **-c** y **-h**, aparecen en la versión de **du** provista por el proyecto GNU. Esto significa que si utilizamos estas últimas en un script desarrollado en Linux e intentamos hacerlo funcionar en FreeBSD o AIX, es muy probable que no funcione como esperamos.

OS derivados de Unix