PRACTICA 1

¿Qué es GNU?

Es un Sistema Operativo tipo Unix (Unix like), pero libre.

- S.O. diseñado por miles de programadores
- S.O. gratuito y de libre distribución (se baja desde la Web, CD, etc.)
- Existen diversas distribuciones (customizaciones)
- Es código abierto, lo que nos permite estudiarlo, personalizarlo, auditarlo, aprovecharnos de la documentación, etc...
- Permite a los usuarios tener la libertad de usar, estudiar, distribuir y mejorar los programas

GNU/Linux al ser un software libre posee las siguientes características:

- Puede ser usado, copiado, estudiado, modificado y redistribuido libremente.
- Generalmente es de costo nulo ← Es un gran error asociar el software libre con el software gratuito ← Pensar en software gratis que se distribuye con restricciones
- Es común que se distribuya junto con su código fuente (código abierto)
- Corrección más rápida ante fallas
- Características que se refieren a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software

Breve historia

Fue iniciado por Richard Stallman en 1983 con el fin de crear un Unix libre (el sistema GNU). En 1985 Richard Stallman creó la Free Software Foundation (FSF o Fundación para el Software Libre) para proveer soportes logísticos, legales y financieros al proyecto GNU. Esta fundación contrato programadores, aunque una porción sustancial del desarrollo fue (y continúa siendo) producida por voluntarios.

En 1990, GNU ya contaba con un editor de textos (Emacs), un compilador (GCC) y gran cantidad de bibliotecas que componen un Unix típico pero faltaba el componente principal, el núcleo (Kernel).

Linus Torvalds ya venía trabajando desde 1991 en un Kernel denominado Linux, el cual se distribuía bajo licencia GPL. Múltiples programadores se unieron a Linus en el desarrollo, colaborando a través de Internet y consiguiendo paulatinamente que Linux llegase a ser un núcleo compatible con UNIX. En 1992, el núcleo Linux fue combinado con el sistema GNU, resultando en un SO libre y completamente funcional. El SO formado por esta combinación es usualmente conocido como "GNU/Linux" o como una "distribución Linux" y existen diversas variantes.

Características más importantes de GNU/Linux:

- Es multiusuario
- Es multitarea y multiprocesador
- Es altamente portable
- Posee diversos intérpretes de comandos, de los cuales algunos son programables
- Permite el manejo de usuarios y permisos

- Todo es un archivo (hasta los dispositivos y directorios)
- Cada directorio puede estar en una partición diferente (/tmp, /home, etc.)
- Es case sensitive
- Es código abierto

¿Qué es POSIX?

POSIX 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.

Es un conjunto de estándares que define varias interfaces de herramientas, comandos y API para Sistemas Operativos similares a UNIX y otros; una norma escrita por la IEEE, que define una interfaz estándar del sistema operativo y el entorno, incluyendo un intérprete de comandos (shell)

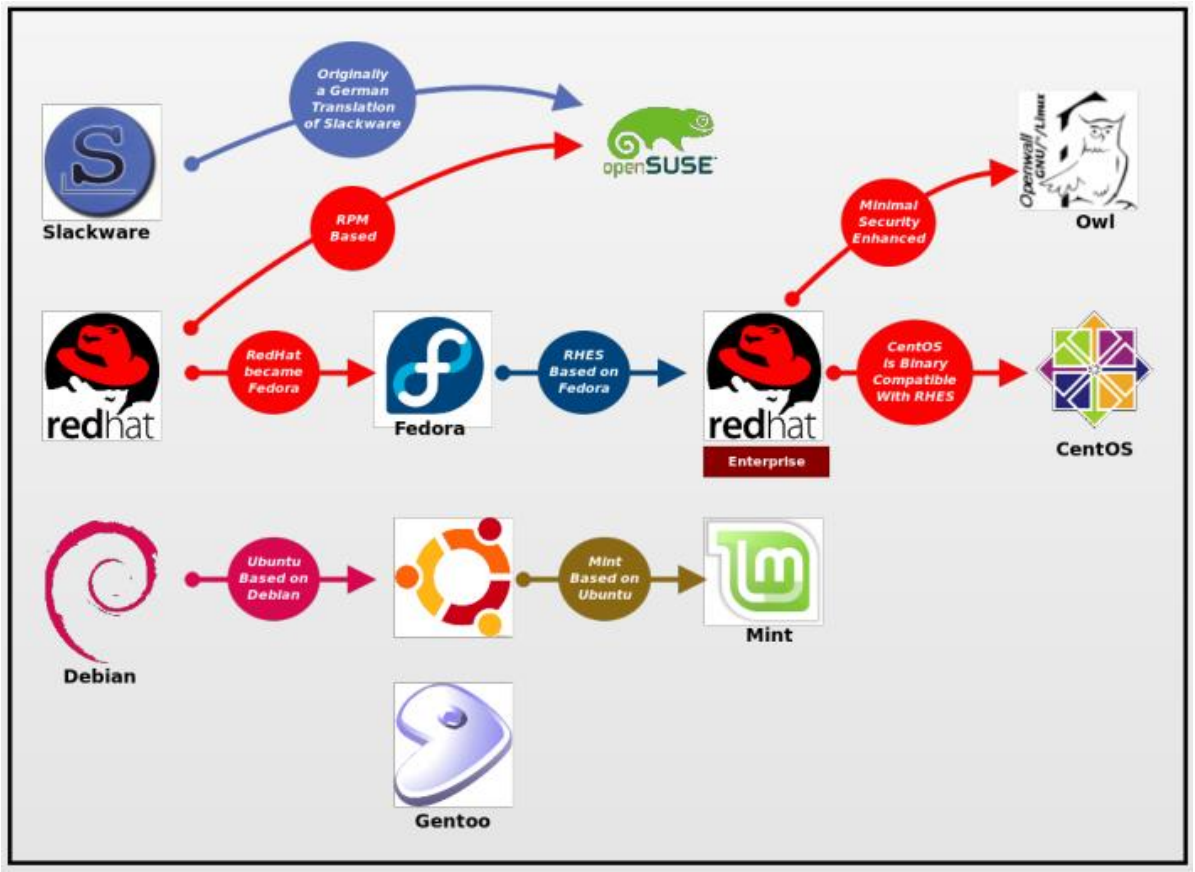
Distribuciones

Son un conjunto de aplicaciones reunidas que permiten brindar mejoras para instalar fácilmente un sistema operativo basado en GNU/Linux. Son "sabores" de GNU/Linux que, en general, se diferencian entre sí por las herramientas para configuración y sistemas de administración de paquetes de software para instalar. La elección de una distribución depende de las necesidades del usuario y de gustos personales.

Una distribución es una customización de GNU/Linux formada por una versión de kernel y determinados programas con sus configuraciones

Existen distribuciones que incluyen paquetes propietarios y otras que son 100% libres.

Las distribuciones de Linux tienen en común el kernel, pero el resto de componentes (las herramientas, la shell, el Display Server, la GUI) varían entre sí, se personalizan o se crean desde cero, por eso las distribuciones son tan diferentes entre sí. Aunque en la mayoría de los casos la principal diferencia es la GUI, o los programas y herramientas que vienen incluidos.



Estructura GNU/Linux

3 componentes fundamentales de GNU/Linux.

- El kernel (núcleo)
- El Shell (interprete de comandos)
- El FileSystem (sistema de archivos)

La estructura básica del S.O es el Kernel que:

- Ejecuta programas y gestiona dispositivos de hardware
- Es el encargado de que el software y el hardware puedan trabajar juntos
- Sus funciones más importantes son la administración de memoria, CPU y la E/S
- En sí, y en un sentido estricto, es el sistema operativo
- Es un núcleo monolítico híbrido:
    - Los drivers y código del Kernel se ejecutan en modo privilegiado
    - Lo que lo hace híbrido es la capacidad de cargar y descargar funcionalidad a través de módulos.
- Esta licenciado bajo la licencia GPL v2

Kernel

El kernel (también conocido como núcleo) es la parte fundamental de un sistema operativo. El kernel o núcleo de linux se podría definir como el corazón de este sistema operativo. Es, a grandes rasgos, el encargado de que el software y el hardware de una computadora puedan trabajar juntos.

La versión estable actual (según la practica) es 5.8.X.

Antes de la serie de Linux 2.6.x, los números pares indicaban la versión "estable" lanzada. Por ejemplo, una para uso de fabricación, como el 1.2, 2.4 ó 2.6. Los números impares, en cambio, como la serie 2.5.x, son versiones de desarrollo, es decir que no son consideradas de producción.

Comenzando con la serie Linux 2.6.x, no hay gran diferencia entre los números pares o impares con respecto a las nuevas herramientas desarrolladas en la misma serie del kernel. Linus Torvalds dictaminó que este será el modelo en el futuro.

Nomenclatura versionado:

A.B.C[.D]

A: Denota versión. Cambia con menor frecuencia.

B: Denota mayor revisión.

C: Denota menor revisión. Solo cambia cuando hay nuevos drivers o características.

D: Cambia cuando se corrige un grave error sin agregar nueva funcionalidad ← Casi no se usa en las ramas 3.x y 4.x, viéndose reflejado en C

¿Es posible tener más de un Kernel de GNU/Linux instalado en la misma máquina?

Se pueden tener varios instalados (Versión anterior o algún kernel virtual), pero en realidad estaría funcionando solo uno, ya que es la parte de un sistema operativo que administra y controla los recursos y procesos

¿Dónde se encuentra ubicado dentro del File System?

En /boot. El primer sector del disco se llama boot sector. Contiene información general de donde se almacena el Kernel y como se arranca

Shell

El Shell (intérprete de comandos) es el programa que recibe lo que se escribe en la terminal y lo convierte en instrucciones para el sistema operativo. Un intérprete de comandos es un programa que lee las entradas del usuario y las traduce a instrucciones que el sistema es capaz de entender y utilizar.

- También conocido como CLI (Command Line Interface).
- Modo de comunicación entre el usuario y el SO.
- Ejecuta programas a partir del ingreso de comandos.

- Cada usuario puede tener una interfaz o Shell. Esta se define por defecto (si no se especifica) pero puede ser personzalida.
    - Un usuario puede cambiar su propio shell a cualquier cosa: la cual, sin embargo, debe estar listada en el archivo/etc/shells.
    - Solo root puede ejecutar un shell que no esté incluido en el archivo/etc/shells.
    - Si una cuenta tiene un shell de inicio de sesión restringido, solo root puede cambiar el shell de ese usuario.
    - Al crear cuentas de usuario con las utilidades useradd o adduser, la marca --shell se puede usar para especificar el nombre del shell de inicio de sesión de un usuario que no sea el especificado en los archivos de configuración respectivos.
- Se pueden personalizar.
- Son programables.
- Bourne Shell (sh), Korn Shell (ksh), Bourne Again Shell (bash)(autocompletado, history, alias).

La shell no forma parte del kernel básico del SO; sino que la misma “dialoga” con el kernel.

No es muy difícil darse cuenta de por qué el shell no es parte del kernel. Como el shell se usa para interpretar las órdenes del usuario y ejecutarlas, si el mismo estuviese en el kernel tendría acceso a instrucciones propias que usa el SO para la gestión de los distintos dispositivos del hardware. Razón por la cual se abstrae al usuario del manejo de dispositivos hardware, dejándolo al kernel.

¿Dónde se ubican (path) los comandos propios y externos al Shell?

El shell nos permite ejecutar:

Comandos externos, por ejemplo: ls, cat, mkdir, etc.

- son programas ajenos al Shell
- cuando se lanzan inician un nuevo proceso
- se buscan en los directorios indicados en la variable PATH

Comandos internos (builtin commands), por ejemplo: cd, bg, alias, eval, exec, pwd, etc.

- se ejecutan en el mismo proceso del shell, sin lanzar un nuevo proceso

La diferencia fundamental es que los internos están incorporados a la consola y se pueden ejecutar directamente, mientras que para los externos hay que indicar la ruta hasta la ubicación del comando.

Para los comandos externos puede ser que no tengamos que indicar la ruta hasta la ubicación del mismo de forma explícita, si esta ruta está incluida en la variable de entorno PATH.

También debemos tener precaución en el caso de que el comando exista tanto de forma interna y externa, ya que las dos versiones del comando pueden dar resultados distintos, por lo que si queremos estar seguros de que estamos ejecutando la versión externa debemos indicar la ruta del comando (p.e. pwd ó /bin/pwd)

Los comandos internos son nativos del shell de linux que estemos usando (bash por ejemplo). Estos se suelen encontrar en el directorio /usr/bin. Los externos se encuentran en la variables $PATH, no son nativos del shell de Linux.

FileSystem

El Filesystem es la forma en que dentro de un SO se organizan y se administran los archivos. Esa administración comprende:

- Métodos de acceso: cómo se acceden los datos contenidos en el archivo.
- Manejo de archivos: cómo actúan los mecanismos para almacenar, referenciar, compartir y proteger los archivos.
- Manejo de la memoria secundaria: Cómo se administra el espacio para los archivos en memoria secundaria.
- Mecanismos de integridad: con qué métodos se garantiza la incorruptibilidad del archivo.

El adoptado por GNU/Linux es el Extended (v2, v3, v4).

Linux no permite ver por defecto el contenido de las particiones de Windows. Para poder hacerlo será necesario que montemos la partición NTFS o FAT en la que está Windows. Aunque en estos momentos existen distribuciones de GNU-Linux que pueden realizar operaciones de lectura y escritura sobre ellas.

La estructura de archivos es una estructura jerárquica en forma de árbol invertido, donde el directorio principal (raíz) es el directorio "/", del que cuelga toda la estructura del sistema. Este sistema de archivos permite al usuario crear, borrar y acceder a los archivos sin necesidad de saber el lugar exacto en el que se encuentran. No existen unidades físicas, sino archivos que hacen referencia a ellas.

/bin: En donde se residen los comandos principales de Linux como ls o mv. Estan los comandos que pueden usar todos los usuarios (incluido el root).

/boot: Aquí se encuentran los cargadores de inicio y los archivos de inicio del sistema.

/dev: En esta ruta se encuentran montados todos los dispositivos físicos como el USBs o el DVDs.

/etc: Contiene la configuración de los paquetes instalados.

/home: Los usuarios del sistema tendrán su carpeta personal para colocar todas sus carpetas adicionales con su nombre como se muestra a continuación: /home/likegeeks.

/lib: Aquí se guardan las librerías de los paquetes instalados ya que estas librerías son compartidas por todos los paquetes. A diferencia de Windows, puedes encontrar duplicados en diferentes carpetas.

/media: En esa ruta se encuentran los dispositivos externos como los DVDs ylos pendrives USB, desde aquí puedes acceder a sus archivos desde aquí.

/mnt: Aquí se montan otras cosas como localizaciones de red y algunas distribuciones que puedas tener montadas en un pendrive o DVD.

/opt: Algunos paquetes opcionales se encuentran aquí y esta ruta es administrada por el administrador de paquetes.

/proc: Debido a que todo en Linux es un archivo, esta es una carpeta que tiene los procesos ejecutándose en el sistema, y puedes acceder a ellos para obtener información acerca de los procesos actuales.

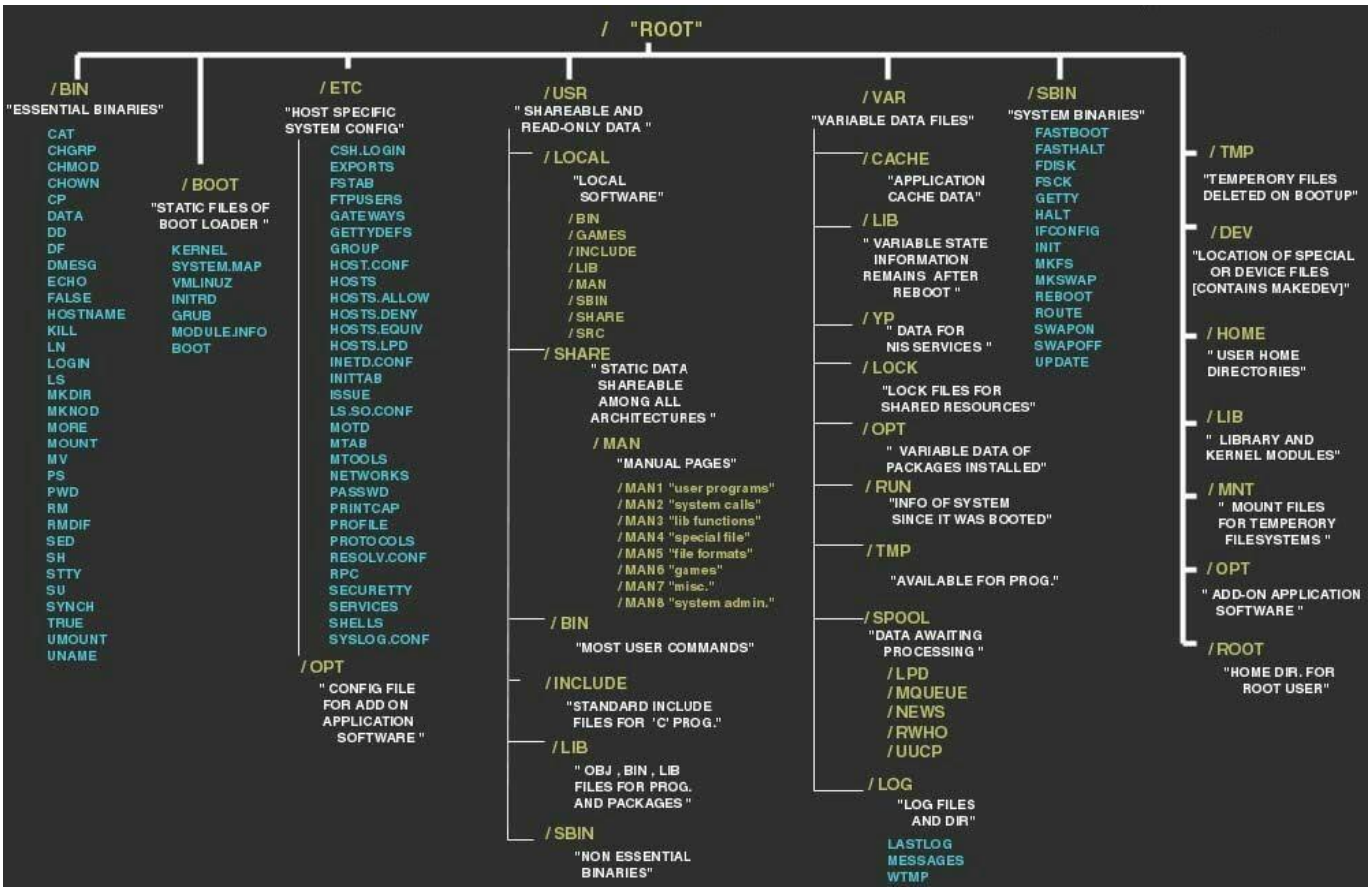
/root: La carpeta home para el usuario root.

/sbin: Como /bin, pero con archivos binarios solo para el usuario root.

/tmp: Contiene los archivos temporales.

/usr: Aquí es donde las utilidades y los archivos se comparten entre usuarios en Linux.

/var: Contiene registros del sistema y otros datos variables



ANOTACION: /dev/null es un archivo especial que descarta toda la información que se escribe en o se redirige hacia él. A su vez, no proporciona ningún dato a cualquier proceso que intente leer de él, devolviendo simplemente un EOF o fin de archivo.

EJEMPLO: "find / –name pepe 2> /dev/null" muestra los archivos llamados pepe, siempre y cuando tenga permisos de acceso a ellos

Directorio	Descripción
/	Es la raíz del sistema de directorios. Aquí se monta la partición principal Linux EXT.
/etc	Contiene los archivos de configuración de la mayoría de los programas.
/home	Contiene los archivos personales de los usuarios.
/bin	Contiene comandos básicos y muchos programas.
/dev	Contiene archivos simbólicos que representan partes del hardware, tales como discos duros, memoria...
/mnt	Contiene subdirectorios donde se montan (se enlaza con) otras particiones de disco duro, CDROMs, etc.
/tmp	Ficheros temporales o de recursos de programas.
/usr	Programas y librerías instalados con la distribución
/usr/local	Programas y librerías instalados por el administrador
/sbin	Comandos administrativos
/lib	Librerías varias y módulos ("trozos") del kernel
/var	Datos varios como archivos de log (registro de actividad) de programas, bases de datos, contenidos del servidor web, copias de seguridad...
/proc	Información temporal sobre los procesos del sistema (explicaremos esto más en profundidad posteriormente).

PARTICIONES

Una partición de disco, en mantenimiento, es el nombre genérico que recibe cada división presente en una sola unidad física de almacenamiento de datos. Toda partición tiene su propio sistema de archivos (formato); generalmente, casi cualquier sistema operativo interpreta, utiliza y manipula cada partición como un disco físico independiente, a pesar de que dichas particiones estén en un solo disco físico.

Ventajas y deventajas:

-Es una buena práctica separar los datos del usuario de las aplicaciones y/o Sistema Operativo instalado

-Tener una partición de Restore de todo el sistema

-Poder ubicar el Kernel en una partición de solo lectura, o una que nisiquiera se monta (no está disponible a los usuarios).

-Particionar demasiado un disco puede tener desventajas. (Al achicar el tamaño del disco rígido físico, puede ocurrir que archivos grandes no entren en el tamaño de una de las particiones nuevas).

Debido al tamaño acotado en el MBR para la tabla de particiones:

• Se restringe a 4 la cantidad de particiones primarias

• 3 primarias y una extendida con sus respectivas particiones lógicas

• Una de las 4 particiones puede ser extendida, la cual se subdivide en volúmenes lógicos

Partición primaria: división cruda del disco (puede haber 4 por disco). Se almacena información de la misma en el MBR.

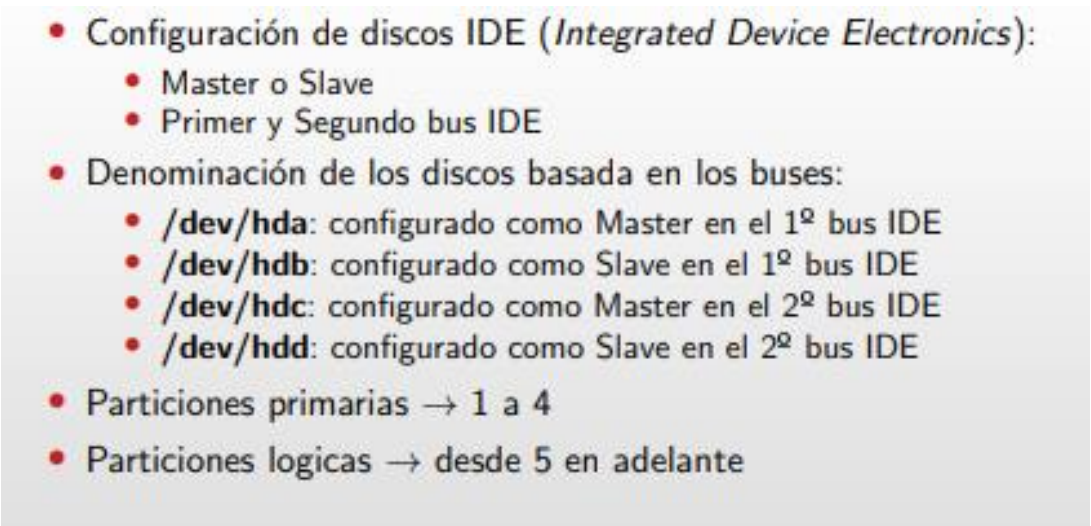
Partición extendida: sirve para contener unidades lógicas en su interior. Solo puede existir una partición de este tipo por disco. No se define un tipo de FS directamente sobre ella.

Partición lógica: ocupa la totalidad o parte de la partición extendida y se le define un tipo de FS. Las particiones de este tipo se conectan como una lista enlazada.

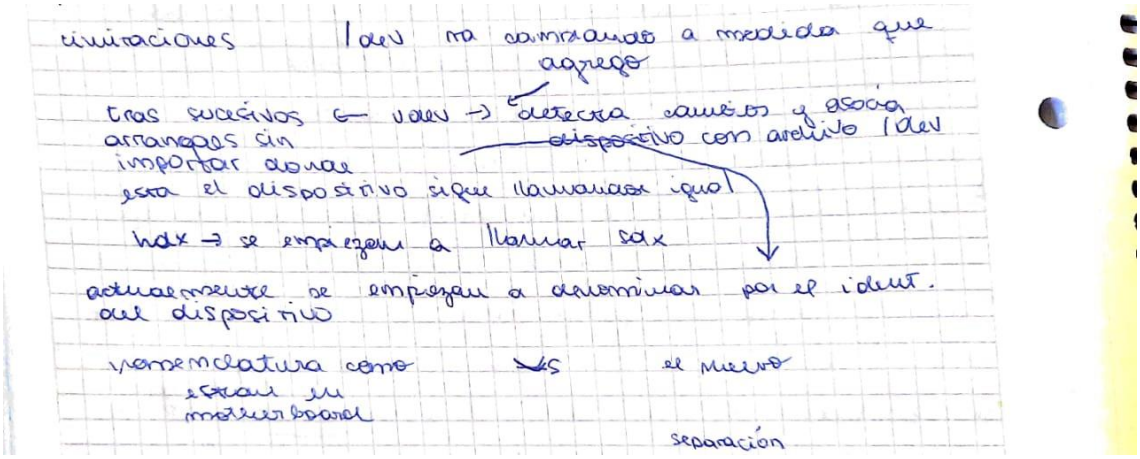
¿Cómo se identifican las particiones en GNU/Linux? (Considere discos IDE, SCSI y SATA).

- Disqueteras
  - Primera disquetera: **/dev/fd0** (en Windows sería la disquetera A:)
  - Segunda disquetera: **/dev/fd1**
- Discos duros (en general: **/dev/hdx#,** donde x es el disco y # es la partición)
  - Primer disco duro: (todo el disco) **/dev/hda**
    - Particiones primarias
      - Primera partición primaria: **/dev/hda1**
      - Segunda partición primaria: **/dev/hda2**
      - Tercera partición primaria: **/dev/hda3**
      - Cuarta partición primaria: **/dev/hda4**
    - Particiones lógicas
      - Primera partición lógica: **/dev/hda5**
      - Sucesivamente: **/dev/hda#**
  - Segundo disco duro: (todo el disco**) /dev/hdb**
    - Particiones primarias
      - Primera partición primaria: **/dev/hdb1**
      - Segunda partición primaria: **/dev/hdb2**
      - Tercera partición primaria: **/dev/hdb3**
      - Cuarta partición primaria: **/dev/hdb4**
    - Particiones lógicas
      - Primera partición lógica: **/dev/hdb5**
      - Sucesivamente: **/dev/hdb#**
- Discos SCSI
  - Primer disco SCSI: **/dev/sda**
  - Segundo disco SCSI: **/dev/sdb**
    - Sucesivamente ...

Primer CD-ROM SCSI: /dev/scd0, también conocido como **/dev/sr0**



- Configuración de discos SCSI: se basa en *LUN*
- Denominación de los discos basada en la identificación de los buses:
  - **/dev/sda**
  - **/dev/sdb**
  - **/dev/sdc**
  - **/dev/sdd**
  - ...
- La nomenclatura para los discos SATA es la misma
- Particiones primarias:
  - Se numeran de la 1 a la 4 (solo estas se pueden marcar como activas → booteables)
- Particiones extendidas:
  - Sus unidades o particiones lógicas se numeran a partir de la 5



Para instalar GNU/Linux como mínimo es necesario una partición (para el /). Es recomendable crear al menos 2 (/ y SWAP). Usualmente se suelen tener tres, una para el sistema/programas (/), otra para los datos (/home) y otra para swap.

Si no has separado /home del resto del sistema, cuando vayas a instalar una nueva versión, perderás toda esa documentación.

Punto de montaje / → Primaria. Sistema de ficheros ext4 transaccional.

¿Qué tipo de software para particionar existe? Menciónelos y compare.

Existen 2 tipos:

- Destructivos: permiten crear y eliminar particiones (fdisk)
- No destructivo: permiten crear, eliminar y modificar particiones (fips, gparted) ← generalmente las distribuciones permiten hacerlo desde la interfaz de instalación

Arranque (bootstrap) de un Sistema Operativo:

En las arquitecturas x86 la BIOS de la motherboardes un chip especial que guarda configuración inicial de la computadora.

Su función principal es la de iniciar los componentes de hardware y lanzar el sistema operativo de un ordenador cuando lo encendemos (a través del MBC). También carga las funciones de gestión de energía y temperatura del ordenador.

Cuando enciendes tu ordenador lo primero que se carga en él es el BIOS. Este firmware entonces se encarga de iniciar, configurar y comprobar que se encuentre en buen estado el hardware del ordenador, incluyendo la memoria RAM, los discos duros, la placa base o la tarjeta gráfica. Cuando termina selecciona el dispositivo de arranque (disco duro, CD, USB etcétera) y procede a iniciar el sistema operativo, y le cede a él el control de tu ordenador.

EFI

EFI es nexo entre el SO y el firmware:

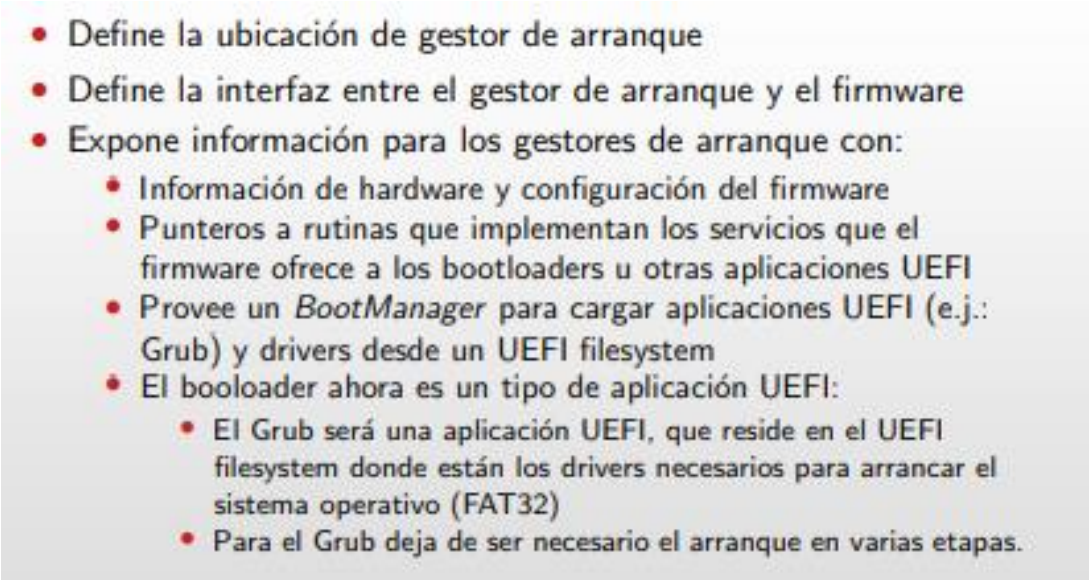
- Utiliza el sistema GPT (GUID partition table) para solucionar limitaciones del MBR, como la cantidad de particiones.
- GPT especifica la ubicación y formato de la tabla de particiones en un disco duro.
- Es parte de EFI. Puede verse como una sustitución del MBR.
- La especificación EFI es propiedad de Intel
- Alternativa para reemplazar la BIOS
- Se pueden crear hasta 128 particiones
- Se mantiene un MBR para tener compatibilidad con el esquema BIOS
- GPT usa un modo de direccionamiento lógico (logical block addressing LBA) en lugar de cylinder-header-sector. GPT puede verse como una sustitución del MBR.
- El MBR “heredado” se almacena en el LBA 0.
- En el LBA 1 está la cabecera GPT. La tabla de particiones en sí está en los bloques sucesivos
- La cabecera GPT y la tabla de particiones están escritas al principio y al final del disco (redundancia)

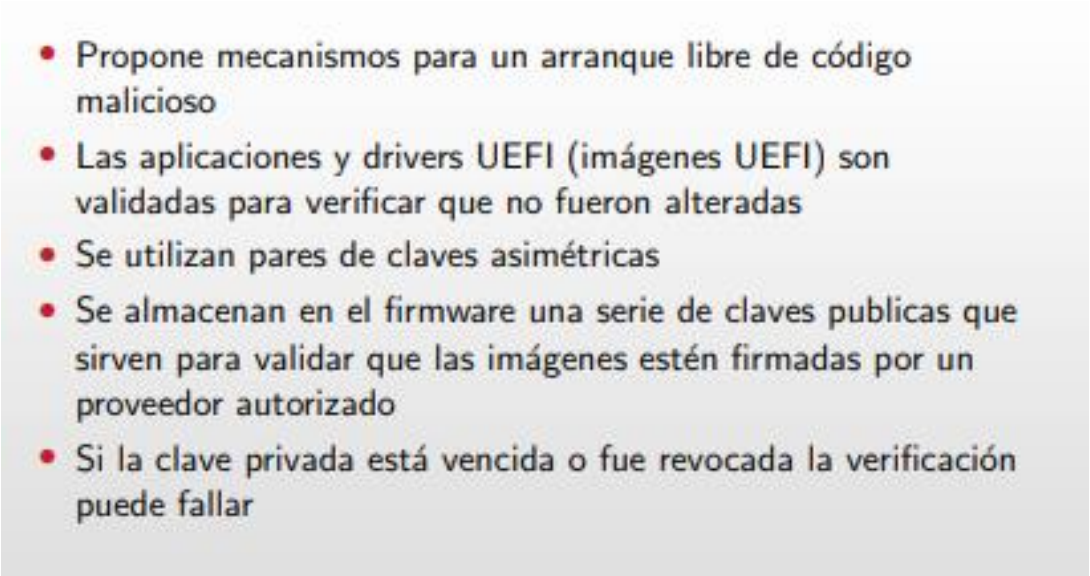
UEFI

A mediados de la década pasada las empresas tecnológicas se dieron cuenta de que el BIOS estaba quedándose obsoleto, y 140 de ellas se unieron en la fundación **UEFI** para renovarla y reemplazarla por un sistema más moderno. En esencia, todo lo que hace el BIOS lo hace también la UEFI. Pero también tiene otras funciones adicionales y mejoras sustanciales, como una interfaz gráfica mucho más moderna, un sistema de inicio seguro, una mayor velocidad de arranque o el soporte para discos duros de más de 2 TB

- Alianza entre varias compañías con el objetivo de modernizar el proceso de arranque.
- UEFI es propiedad del UEFI Forum.
- Define la interfaz entre el gestor de arranque y el firmware más sencilla y fácil de comprender.
- Aporta mayor velocidad en el arranque de los equipos

- UEFI mejora la seguridad con su funcionalidad Secure Boot





El MBR (master boot record) es el primer sector del disco (cilindro 0, cabeza 0, sector 1). Esto se carga a memoria y se ejecuta. Es un registro de arranque principal que puede contener un código de arranque denominado MBC (master boot code) y una marca de 2 bytes que indica su presencia o puede solamente contener la tabla de particiones. En el último caso el BIOS ignora este MBR.

Existe un MBR en todos los discos y si existiese más de un disco rígido en la máquina, sólo uno es designado como Primary Master Disk. El tamaño del MBR coincide con el tamaño estándar de sector: 512 bytes.

El MBC es un pequeño código que permite arrancar el SO. La última acción del BIOS es leer el MBC, lo lleva a memoria y lo ejecuta.

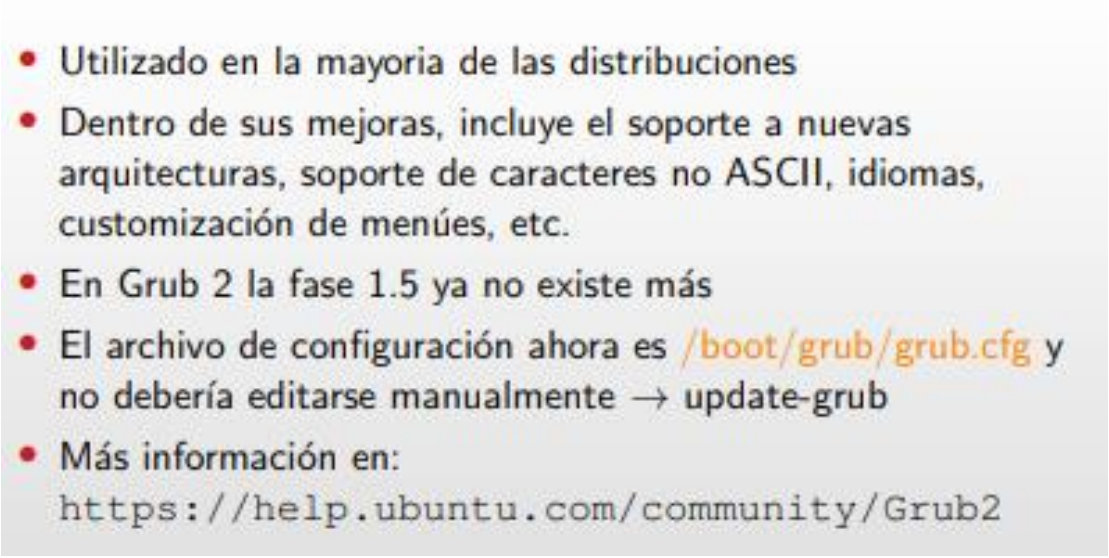
- El tamaño del MBR coincide con el tamaño estándar de sector, 512 bytes:
  - Los primeros bytes corresponden al *Master Boot Code* (MBC)
  - A partir del byte 446 está la tabla de particiones. Es de 64 bytes
  - Al final existen 2 bytes libres o para firmar el MBR
- Es creado con algún utilitario
- El MBC es un pequeño código que permite arrancar el SO
- La última acción del *BIOS* es leer el MBC. Lo lleva a memoria y lo ejecuta
- Si se tiene un sistema instalado → bootloader de MBC típico. Sino → uno diferente (*multietapa*)

El bootloader o cargador de arranque es un programa que permite cargar el Sistema Operativo. Puede llegar a cargar un entorno previo a la carga del sistema

- Generalmente se utilizan los cargadores multietapas, en los que varios programas pequeños se van invocando hasta lograr la carga del SO
- En cierto sentido, el código del BIOS/UEFI forma parte del bootloader, pero el concepto esta más orientado al código que reside en el Master Boot Record (512b)
- El MBR esta formado por el MBC (446b) y la Tabla de Particiones (64b)
- Solo el MBC del Primary Master Disk es tenido en cuenta
- El MBR existe en todos los discos, ya que contiene la tabla de particiones

- **GRand Unified Bootloader**: gestor de arranque múltiple más utilizado
- En el MBR solo se encuentra la fase 1 del que solo se encarga de cargar la fase 1.5
- Fase 1.5: ubicada en los siguientes 30 KB del disco (*MBR gap*). Carga la fase 2
- Fase 2: interfaz de usuario y carga el Kernel seleccionado
- Se configura a través del archivo /boot/grub/menu.lst
- Algunas líneas:
  - default: define el SO por defecto a bootear
  - timeout: tiempo de espera para cargar el SO por defecto

```
title Debian GNU/Linux
root (hd0,1) #(Disco,Particion)
kernel /vmlinuz-2.6.26 ro quiet root=/dev/hda3
initrd /initrd-2.6.26.img
```



Entonces:

- 1. Se ejecuta el código de la BIOS

- 2. El hardware lee el sector de arranque

- 3. Se carga el gestor de arranque

- 4. Se carga el kernel

¿Es posible tener en una PC GNU/Linux y otro Sistema Operativo instalado? Justifique.

Si, ya que un disco rígido puede particionarse y en cada partición tener un sistema de archivos distinto (partición primaria), sería como tener varios discos distintos, uno con cada SO, por lo tanto, se necesitará un gestor de arranque como los descriptos arriba.

Archivos

Como en Windows, se puede emplear un cierto criterio de "tipo" para marcar las distintas clases de ficheros empleando una serie de caracteres al final del nombre que indiquen el tipo de fichero del que se trata. Así, los ficheros de texto, HTML, las imágenes PNG o JPEG tienen extensiones .txt, .htm (o .html), .png y .jpg (o .jpeg) respectivamente. Pese a esto Linux sólo distingue tres tipos de archivos:

- Archivos o ficheros ordinarios, son los mencionados anteriormente.
- Directorios (o carpetas), es un archivo especial que agrupa otros ficheros de una forma estructurada.
- Archivos especiales, son la base sobre la que se asienta Linux, puesto que representan los dispositivos conectados a un ordenador, como puede ser una impresora. De esta forma introducir información en ese archivo equivale a enviar información a la impresora. Para el usuario estos dispositivos tienen el mismo aspecto y uso que los archivos ordinarios

Comandos

Cat: El comando ‘cat’ imprimirá por pantalla el contenido del fichero sin ningún tipo de paginación ni posibilidad de modificarlo. Básicamente concatena archivos o la salida estándar en la salida estándar. Podemos pasarle parámetros como

More: Al igual que ‘cat’, ‘more’ permite visualizar por pantalla el contenido de un fichero de texto, con la diferencia con el anterior de que ‘more’ página los resultados. Primero mostrará por pantalla todo lo que se pueda visualizar sin hacer scroll y después, pulsando la tecla espacio avanzará de igual modo por el fichero.

Less: El comando ‘less’ es el más completo de los tres, pues puede hacer todo lo que hace ‘more’ añadiendo mayor capacidad de navegación por el fichero (avanzar y retroceder) además de que sus comandos están basados en el editor ‘vi’, del cual se diferencia en que no tiene que leer todo el contenido del fichero antes de ser abierto. Tiene una gran cantidad de opciones y parámetros, como siempre lo recomendable:

(a) shutdown

El comando shutdown se utiliza para apagar o reiniciar Linux desde la terminal.

Detener el sistema de forma segura.

shutdown [OPTIONS] [TIME] [MESSAGE]

(b) reboot

Reinicia SO

-reboot, -r: reinicia el sistema

(c) halt

El comando halt se utiliza para apagar el ordenador

halt [-d | -f | -h | -n | -i | -p | -w]

reboot [-d | -f | -i | -n | -w]

poweroff [-d | -f | -h | -n | -i | -w]

OPCIONES:

- d	**No escribir registro wtmp (en el archivo /var/log/wtmp) El flag -n implica -d**
- h	Poner todos los discos duros del sistema en modo de espera antes de que el sistema se detenga o apague
- n	No sincronizar antes de reiniciar o detener
- i	Apagar todas las interfaces de red.
- p	Cuando detenga el sistema, lo apaga también. Esto es por defecto cuando el halt se llama como poweroff.
- w	No reiniciar o detener, sólo escribir el registro wtmp (en el archivo /var/log/wtmp)

(d) locate

El comando locate es una alternativa útil, ya que es más rápido que find para realizar búsquedas. Eso se debe a que sólo escanea tu base de datos de Linux en lugar de todo el sistema

locate [my-file]

(e) uname

Si utilizamos el comando sin argumentos nos entregara la palabra Linux extraída de la información del Kernel.

$ uname

Si deseamos extraer la información de la versión del kernel utilizamos el parámetro -r.

$ uname -r

Si deseamos extraer la fecha de cuando la versión del kernel fue liberada utilizamos el parámetro -v.

$ uname -v

(f) dmesg

Se usa para examinar o controlar el ring buffer del kernel. La acción predeterminada es mostrar todos los mensajes del ring buffer.

Debido a toda la información desplegada, es difícil llevar a cabo alguna tarea de administración allí. Podemos hacer uso del parámetro "-H" con el fin de indicarle a dmesg que la salida sea legible para los usuarios, lo cual simplificará las tareas de soporte. Allí encontramos detalles mucho más claros sobre el anillo del kernel.

Otra alternativa para realizar un análisis con dmesg es con el parámetro "-w", el cual nos permite escribir un script para analizar el resultado usando una expresión regular con el fin de filtrar los eventos para su posterior análisis:

(g) lspci

lspci es un comando para los sistemas operativos Unix-like que imprime listas con información detallada sobre todos los Buses y dispositivos del sistema

(h) at

Permite programar tareas únicas en nuestro sistemas GNU/Linux.

N os permite programar tareas para que se ejecuten a determinada fecha y hora .

at [hora] [fecha]

El comando at, nos puede ser útil para apagar el sistema a una hora especifica, realizar una copia de seguridad única, enviar un correo electrónico como recordatorio a la hora especificada, entre otras muchas cosas.

(i) netstat

netstat (estadísticas de red) es una herramienta de línea de comandos que muestra las conexiones de red (entrantes y salientes), tablas de enrutamiento y una serie de estadísticas de interfaz de red.

(j) mount

El mandato mount ordena al sistema operativo que haga que un sistema de archivos esté disponible para su utilización en una ubicación determinada (el punto de montaje).

(k) umount

Este comando permite desmontar un sistema de archivos montado previamente.El uso del comando umount garantiza que toda la información mantenida en memoria por el sistema operativo se escriba en el dispositivo antes de desmontarlo

Sintaxis: umount dispositivo | punto_montaje

(l) head

El comando head muestra de modo predeterminado las diez primeras líneas de un archivo. Se puede modificar esta opción a las N primeras líneas del archivo con la sintaxis head -nN.

(m) losetup

se usa para asociar loop devices con archivos regulares o block devices, también para desacoplar loop devices, y para hacer queries del status de un loop device. (dispositivo iterador = loop device)

(n) write

El comando write permite mandar un mensaje a otro usuario del sistema especificando como parámetros el usuario al que enviar el mensaje y la TTY asociada:

$ write usuario tty.

Para finalizar la escritura del mensaje y enviarlo presionamos CTRL + D. La TTY se especificará cuando el usuario al que enviemos el mensaje tenga más de una sesión abierta. En el caso de que no le especificamos, se enviará automáticamente a la tty del usuario con actividad más reciente

(ñ) mkfs

se utiliza para dar formato a un dispositivo de almacenamiento de bloque con un determinado sistema de archivos.

(o) fdisk (con cuidado)

es una utilidad de línea de comandos basada en texto para ver y administrar particiones de disco duro en Linux. Con fdisk puedes ver, crear, cambiar el tamaño, eliminar, cambiar, copiar y mover particiones.más info de Fdisk: https://maslinux.es/comando-fdisk-para-administrar-particiones-de-disco-en-gnulinux/

EMULADORES/VIRTUALIZADORES

- Básicamente se pueden considerar 3 tipos:
    - Emulación:
        - Emulan hardware
        - Tienen que implementar todas las instrucciones de la CPU
        - Es muy costosa y poco eficiente

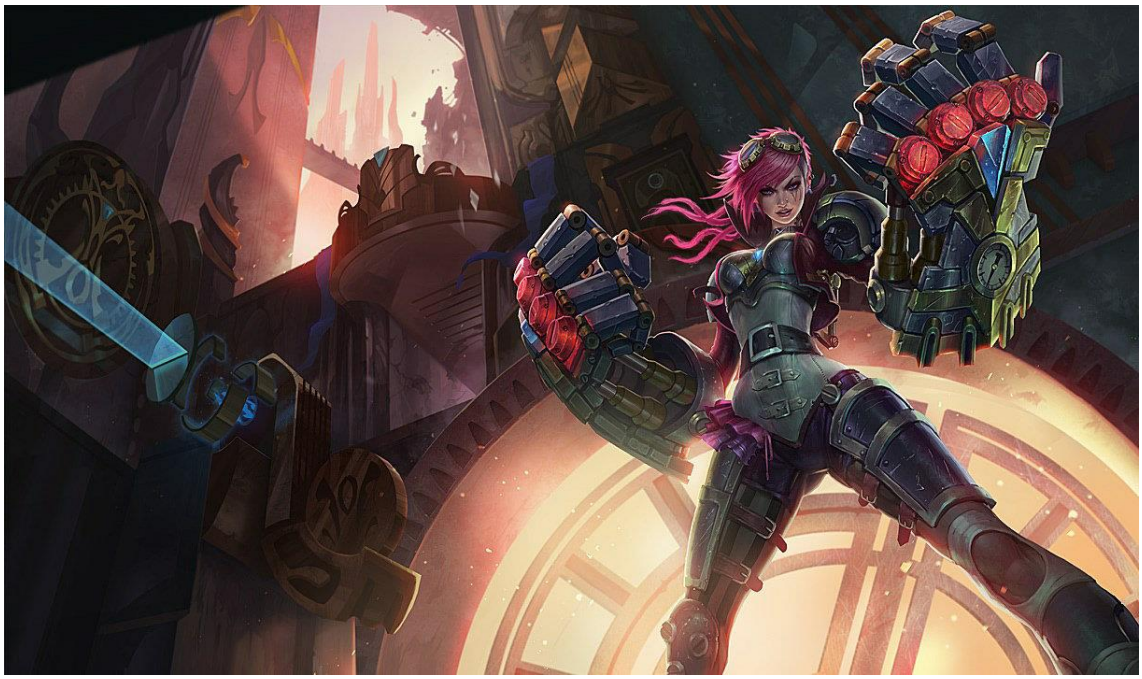
        - Permite ejecutar arquitecturas diferentes a las soportadas por el hardware
- Virtualización completa:
    - Permiten ejecutar SO huéspedes en un sistema anfitrión (host)
    - Utilizan en el medio un hypervisor o monitor de máquinas virtuales
    - El SO huésped debe estar soportado en la arquitectura anfitriona
    - Es más eficiente que la emulación (Intel-VT y AMD-V)
- Paravirtualización:
    - Permite correr SOs modificados exclusivamente para actuar en entornos virtualizados
    - Mayor eficiencia que la virtualización

Las principales diferencias entre ellos son:

- Los virtualizadores aprovechan el CPU sobre la que están trabajando, lo cual los hace más veloces.
- En un emulador se puede correr cualquier arquitectura. En un virtualizador solo se puede correr la arquitectura virtualizada

PRACTICA 2

VI



Editor de textos **vim**

- Presente en cualquier distribución de GNU/Linux
- Posee 3 modos de ejecución:
  - Modo Insert (**Ins** o **i**)
  - Modo Visual (**v**)
  - Modo de Órdenes o Normal (**Esc**)
- Se le puede enviar una serie de comandos útiles
  - **w**: escribir cambios
  - **q** ó **q!**: salir del editor
  - **dd**: cortar
  - **y**: copiar al portapapeles
  - **p**: pegar desde el portapapeles
  - **u**: deshacer
  - /frase: busca "frase" dentro del archivo

Proceso de Arranque SystemV *relacionado con pagina 13 (volver)

Pasos del proceso de inicio de un sistema GNU/Linux, desde que se prende la PC hasta que se logra obtener el login en el sistema.

1. Se empieza a ejecutar el código del BIOS
2. El BIOS ejecuta el POST
3. El BIOS lee el sector de arranque (MBR)
4. Se carga el gestor de arranque (MBC)
5. El bootloader carga el *kernel* y el *initrd*
6. Se monta el initrd como sistema de archivos raíz y se inicializan componentes esenciales (ej.: scheduler)
7. El Kernel ejecuta el proceso *init* y se desmonta el initrd
8. Se lee el /etc/inittab
9. Se ejecutan los scripts apuntados por el *runlevel 1*
10. El final del runlevel 1 le indica que vaya al runlevel por defecto
11. Se ejecutan los scripts apuntados por el runlevel por defecto
12. El sistema está listo para usarse

Proceso INIT

Lo ejecuta el Kernel y su función es cargar todos los subprocesos necesarios para el correcto funcionamiento del SO. Posee PID 1 y se encuentra en /sbin/init. Se lo configura a través del archivo/etc/inittab. No tiene padre y es padre de todos los procesos. Es el encargado demontar los filesystems y de hacer disponibles los demás dispositivos.

pstree: muestra los procesos en ejecución en forma de árbol (padre e hijo) siendo systemd el padre de todos los procesos.

Runlevels

- Es el modo en que arranca linux (3 en redhat, 2 en Debian)
- Un nivel de ejecución es básicamente una configuración de programas y servicios que se ejecutarán orientados a un determinado funcionamiento.
- El proceso de arranque se divide en niveles. Cada uno es responsable de levantar o bajar una serie de servicios.
- Se encuentran definidos en /etc/inittab: Es el archivo de configuración de init. Cuando el sistema se arranca, se verifica si existe un runlevel predeterminado en el archivo/etc/inittab, si no, se debe introducir por medio de la consola del sistema. Después se procede a ejecutar todos los scripts relativos al runlevel especificado.
    - Formato:
      id:niveles_ejecucion:acción:proceso
        - Id: identifica la entrada en inittab (1 a 4 caracteres)
        - Niveles_ejecucion: el/los nivel de ejecución en los que se realiza la acción
        - Acción: describe la acción a realizar
          -wait: se inicia cuando se entra al runlevel e init espera a que termine
          - initdefault

- ctrlaltdel: se ejecutará cuando init reciba la señal SIGINT
- off, repawn, once, boot, bootwait, powerwait, otras...

- Proceso: el proceso exacto que será ejecutado

Existen 7 runlevels, y permiten iniciar un conjunto de procesos al arranque o apagado del sistema. Según el estándar:

0. halt (parada)
1. single user mode (monousuario)
2. multiuser, without NFS (modo multiusuario sin soperte de red)
3. full multiuser mode console (modo multiusuario completo por consola)
4. no se utiliza
5. X11 (modo multiusuario completo con login grafico basado en X)
6. reboot

En el caso del modelo runlevel de SystemV, cuando el proceso init arranca, utiliza un fichero de configuración llamado /etc/inittab para decidir el modo de ejecución en el que va a entrar. En este fichero se define el runlevel por defecto (initdefault) en arranque (por instalación en Fedora el 5, en Debian el 2), y una serie de servicios de terminal por activar para atender la entrada del usuario. Después, el sistema, según el runlevel escogido, consulta los ficheros contenidos en /etc/rcn.d, donde n es el número asociado al runlevel (nivel escogido), en el que se encuentra una lista de servicios por activar o parar en caso de que arranquemos en el runlevel, o lo abandonemos. Dentro del directorio encontraremos una serie de script so enlaces a los scripts que controlan el servicio. Cada script posee un nombre relacionado con el servicio, una S o K inicial que indica si es el script para iniciar (S) o matar (K) el servicio, y un número que refleja el orden en que se ejecutarán los servicios.

No todas las distribuciones respetan los estándares.

Comando *init* <x> para cambiar de runlevel

SCRIPTS RC

Los scripts RC se encargan de cargar o cerrar los servicios necesarios para que el sistema funcione, de acuerdo con el runlevel que se está iniciando. Estos servicios se encuentran en /etc/init.d/.Sin embargo, no todos los servicios se cargan en todos los runlevels.

Los servicios a cargar se encuentran en el directorio /etc/rcX.d/, donde X es el runlevel a cargar. En realidad, en estos directorios no hay más que enlaces simbólicos a /etc/init.d/ (referencias). Los nombres en estos directorios empiezan con una letra (S o K) seguidos de un número (de 2 dígitos que indican el orden en el que se arrancará el servicio) y el nombre del servicio.

/etc/rc.d/rc cuando entra en un determinado nivel de ejecución realiza las siguientes acciones:

1.Ejecuta, por orden de nombre, todos los scripts que comienzan por K en el directorio correspondiente al nivel, utilizando como argumento para dicho script la opción stop.

2.Ejecuta, por orden de nombre, todos los scripts que comienzan por S en el directorio correspondiente al nivel, utilizando como argumento para dicho script la opción start.

- Los scripts que se ejecutan están en /etc/init.d
- En /etc/rcX.d (donde X = 0..6) hay links a los archivos del /etc/init.d
- Formato de los links: [S|K]<orden><nombreScript>

```
$ ls -1 /etc/rcS.d/
S55urandom
S70x11-common
```

- **S**: lanza el script con el argument start
- **K**: lanza el script con el argument stop

Insserv

El programa insserv se usa para administrar el orden de los enlaces simbólicos que están en «/etc/rcX.d/», resolviendo las dependencias de forma automática. Mejora la performance del arranque en sistemas multiprocesadores.

Lo que hace es analizar automáticamente los datos introducidos en la cabecera del script init y guarda los enlaces para los scripts de arranque y parada en los directorios de niveles de ejecución respectivos. Esto permite a cada mantenedor de paquetes especificar en su script de init.d la relación con otros scripts y poder detectar y evitar bucles de dependencias entre scripts así como asegurarse de que todos los scripts se inician en el orden pretendido.

Upstart

- reemplazo basado en eventos, y no en niveles.
- Los servicios se pueden levantar o desactivaren respuesta a ciertos eventos, y este procedimiento permite por ejemplo manejar el reinicio de servicios que mueren de forma inesperada.
- Opera asíncronamente: las tareas y servicios son ejecutados ante eventos (arranque del equipo o inserción de un dispositivo USB) definidos como tareas o Jobs.
- Los jobs se almacenan en el directorio /etc/init.
    - Son scripts en texto plano que definen las acciones a ejecutar. Es compatible con el System V.
    - Dos tipos:
        - Task: ejecución finita (task) → not respawning → exit 0 o uso de stop
        - Service: ejecución indeterminada → respawning

Una de las principales diferencias entre System V y Upstart es que el primero trabaja de forma síncrona mientras que Upstart lo hace de forma asíncrona, es decir, no arranca/para un

servicio después de otro sino que puede hacerlo en paralelo. También gestiona las tareas y servicios de inicio cuando el sistema arranca y los detiene cuando el sistema se apaga.

El demonio init tradicional (SystemV) es estrictamente síncrono, bloqueando futuras tareas hasta que la actual se haya completado. Sus tareas deben ser definidas por adelantado, y solo pueden ser ejecutadas cuando el demonio init cambia de estado

Los jobs son ejecutados ante eventos (arranque del equipo, inserción de un dispositivo USB, etc.):

- Es posible crear eventos pero existen algunos de manera estándar[2]
- Definido por **start on** y **stop on**

Es compatible con SystemV → /etc/init/rc-sysinit.conf, runlevels, scripts en /etc/init.d, objetivo start y stop

Cada job posee un objetivo (*goal start/stop*) y un estado (*state*)

- En base a ellos se ejecuta un proceso específico
- Al inicio, init emite el evento *startup*

- Un job puede tener uno o varias tareas ejecutables como parte de su ciclo de vida y siempre debe existir la tarea principal
- Las tareas de un job se definen mediante *exec* o *script ... end script*
- A través de *initctl* podemos administrar los jobs del demonio de Upstart:
  - start <job>: cambia el objetivo a start del job especificado
  - stop <job>: cambia el objetivo a stop del job especificado
  - emit <event>: event es emitido causando que otros jobs cambien a objetivo start o stop
- No más /etc/inittab

REEMPLAZO SCRIPTS RC EN UPSTART

En concreto Upstart ignora el archivo /etc/inittab, en su lugar proporciona un conjunto integrado de scripts de texto plano de arranque que se encuentran en el directorio /etc/init, con nombre 'servicio'.conf, donde servicio es el programa que init tratará como un job. Los scripts de Upstart ofrecen más acciones que los de SysV, por ejemplo iniciar un servicio siempre que se conecte un determinado dispositivo de hardware.

Systemd

Es un sistema que centraliza la administración de demonios y librerías del sistema

- Mejora el paralelismo de booteo
- Realiza activacion por socket ya que no todos los servicios que se inician en el booteo se utilizan
- Puede ser controlado por systemctl
- Compatible con SysV → si es llamado como init
- El demonio systemd reemplaza al proceso init → este pasa a tener PID 1
- Los runlevels son reemplazados por targets
- Al igual que con Upstart el archivo /etc/inittab no existe más

Las unidades de trabajo (dos estados active o inactive) son denominadas units de tipo:

- Service: controla un servicio particular (.service)
- Socket: encapsula IPC, un sockect del sistema o file system FIFO (.socket) → sockect-based activation
- Target: agrupa units o establece puntos de sincronizaci´on durante el booteo (.target) → dependencia de unidades
- Snapshot: almacena el estado de un conjunto de unidades que puede ser restablecido más tarde (.snapshot)

¿A qué hace referencia el concepto de activación de socket en systemd?

- No todos los servicios que se inician en el booteo se utilizan:
  - impresora
  - servidor en el puerto 80
  - etc.
- Es un mecanismo de iniciación bajo demanda → podemos ofrecer una variedad de servicios sin que realmente esten iniciados
- Cuando el sockect recibe una conexión spawnea el servicio y le pasa el socket
- No hay necesidad de definir dependencias entre servicios → se inician todos los sockets en primer medida

¿A qué hace referencia el concepto de cgroup?

- Permite organizar un grupo de procesos en forma jerárquica
- Agrupa conjunto de procesos relacionados (por ejemplo, un servidor web Apache con sus dependientes)
- Tareas que realiza:
  - Tracking mediante subsistema cgroups → no se utiliza el PID → doble *fork* no funciona para escapar de systemd
  - Limitar el uso de recursos
  - etc.

Es una característica del Kernel de Linux que limita, explica y aísla el uso de recursos (CPU, memoria, E / S de disco, red, etc.) de una colección de procesos.

Usuarios:

UID: USER ID

GID: GROUP ID

Si ponemos dos UID iguales a distinto usuario, el S.O. los tomará como una sola cuenta.

En versiones anteriores podía haber varios usuarios con el mismo UID, luego esto fue cambiado que no de no hacerlo se podía modificar el UID de un usuario a 0 y así este tendrá los mismos permisos que el root

ROOT es el administrador. Sólo hay un usuario root, pero pueden haber más de un usuario administrador.

Ser usuario root en un sistema Unix significa tener acceso al directorio raíz, ese donde tenemos instalado todo el sistema operativo. También se le llama ser un Superusuario y en definitiva es tener acceso total al sistema. Algo parecido a tener derechos de administrador en un sistema Windows.

Solo puede haber 1 usuario root pero pueden haber varios admins

UID = 0

- Todo usuario debe poseer credenciales para acceder al sistema
  - root: es el administrador del sistema (superusuario)
  - otros: usuarios estándar del sistema (/etc/sudoers)
- Archivos de configuración:
  - **/etc/passwd**

```
$ cat /etc/passwd
ndelrio:x:2375:500:Nico del Rio,,,,Usuarios:/
    home/admins/ndelrio:/bin/bash
```

  - **/etc/group**

```
$ cat /etc/group
infraestructura:x:500:
```

  - **/etc/shadow**

```
$ cat /etc/shadow
ndelrio:$1$HamkgCYM$TtgfLJLplItxutaiqh/u9
    /:13273:0:99999:7:::
```

- su: permite usar el intérprete de comandos de otro usuario sin necesidad de cerrar la sesión
- groupadd nombre_grupo : crea un grupo.
- who: dice el usuario activo en ese momento
- useradd <nombreUsuario>:
  - Agrega el usuario
  - Modifica los archivos /etc/passwd
  - Alternativa → adduser
- passwd <nombreUsuario>:
  - Asigna o cambia la contraseña del usuario
  - Modifica el archivo /etc/shadow
- usermod <nombreUsuario>:

  -g: modifica grupo de login (Modifica /etc/passwd)

  -G: modifica grupos adicionales (Modifica /etc/group)

  -d: modifica el directorio home (Modifica /etc/passwd)

- userdel <nombreUsuario>: elimina el usuario
- groupdel <nombreGrupo>: elimina el grupo

PERMISOS

Los permisos están divididos en tres tipos: lectura, escritura y ejecución. Estos permisos pueden ser fijados para tres clases de usuarios: el propietario del archivo o directorio, los integrantes del grupo al que pertenece y todos los demás usuarios.

- Se aplican a directorios y archivos
- Existen 3 tipos de permisos y se basan en una notación octal:

Permiso	Valor	Octal
Lectura	R	4
Escritura	W	2
Ejecución	X	1

- Se aplican sobre los usuarios:
  - Usuario: permisos del dueño → **U**
  - Usuario: permisos del grupo → **G**
  - Usuario: permisos de otros usuario → **O**
- Se utiliza el comando **chmod**:

```
$ chmod 755 /tmp/script
```

- r – permiso de lectura (permite abrirlo, copiarlo, etc).
- w – permiso de escritura (permite modificarlo, borrarlo, etc).
- x – permiso de ejecución (si es binario, permite ejecutarlo).

chmod: es un comando que permite modificar y especificar quien puede manejar el archivo y cómo puede hacerlo.

chmod +rwx softzone -> añade permiso de lectura, escritura y ejecución al propietario.

chmod g+rwx softzone -> añade permiso de lectura, escritura y ejecución al grupo del usuario propietario.

chmod o+rwx softzone -> añade permiso de lectura, escritura y ejecución al resto de usuarios.

También podemos usar el código numérico para cambiar el nivel de permisos de cualquier archivo de la siguiente manera:

chmod 777 softzone -> permiso rwx para propietario, grupo y resto de usuarios.

chmod 700 softzone -> permiso rwx para propietario, y grupo y usuarios sin permisos.

chmod 327 softzone – permiso de escritura y ejecución para propietario, escritura para grupo y rwx para resto de usuarios.

chown: es un comando que permite cambiar el propietario de un archivo o directorio en GNU. Se puede especificar el nombre de usuario o un GID.

chown nombreUsuario archivo1 [archivo2 archivo3…]

chown -R nombreUsuario nombreDirectorio (de manera recursiva la r)

chgrp: es un comando que permite cambiar el grupo de usuarios de un archivo o directorio en GNU. Normalmente este tipo de tareas de asignación de permisos se puede realizar con el comando chown pero chgrp maneja una sintaxis más simple para esta tarea

chgrp groupname archivo

Número	Binario	Lectura (r)	Escritura (w)	Ejecución (x)
0	000	✗	✗	✗
1	001	✗	✗	✓
2	010	✗	✓	✗
3	011	✗	✓	✓
4	100	✓	✗	✗
5	101	✓	✗	✓

6	110	✔	✔	✘
7	111	✔	✔	✔

¿Existe la posibilidad de que algún usuario del sistema pueda acceder a determinado archivo para el cual no posee permisos? Nombrelo, y realice las pruebas correspondientes.

Si, el usuario root, ya que tiene acceso administrativo al sistema (posee permisos de superusuario). Los usuarios normales no tienen este acceso por razones de seguridad.

Los usuarios con acceso a sudo tambien pueden acceder con o sin permisos ya que este comando p permisos de kernel/superusuario

“full path name” es la ruta desde el directorio raiz (root) "/" y **“relative path name”** es la ruta iniciando desde la carpeta desde donde "se esta parado"

Artículo	Descripción
nombre de ruta absoluto	Rastrea la ruta desde el directorio /(raíz). Los nombres de ruta absolutos siempre comienzan con el símbolo de barra inclinada (/).
nombre de ruta relativo	Rastrea la ruta desde el directorio actual a través de su padre o sus subdirectorios y archivos.

Un nombre de ruta absoluto representa el nombre completo de un directorio o archivo desde el directorio /(raíz) hacia abajo. Independientemente de dónde esté trabajando en el sistema de archivos, siempre puede encontrar un directorio o archivo especificando su nombre de ruta absoluto. Ruta absoluta los nombres comienzan con una barra inclinada (/), el símbolo que representa el directorio raíz. El nombre de ruta /A/D/9 es el nombre de ruta absoluto para 9. La primera barra inclinada (/) representa el directorio /(raíz) , que es el lugar de inicio de la búsqueda El resto del nombre de la ruta dirige la búsqueda a A , luego a D y finalmente a 9 .

Pueden existir dos archivos llamados 9 porque los nombres de ruta absolutos a los archivos dan a cada archivo un nombre único dentro del sistema de archivos.Los nombres de ruta /A/D/9 y /C/E/G/9 especifican dos archivos únicos llamados 9 .

A diferencia de los nombres de rutas completas, los nombres de rutas relativas especifican un directorio o archivo basado en el directorio de trabajo actual. Para los nombres de rutas relativas, puede usar la notación punto punto (..) para moverse hacia

arriba en la jerarquía del sistema de archivos. El punto punto (. .) representa el directorio principal. Debido a que los nombres de rutas relativas especifican una ruta que comienza en el directorio actual, no comienzan con una barra inclinada (/). Los nombres de rutas relativas se usan para especificar el nombre de un archivo en el directorio actual o la ruta nombre de un archivo o directorio por encima o por debajo del nivel del directorio actual en el sistema de archivos. Si D es el directorio actual, el nombre de la ruta relativa para acceder a 10 es F/10 . Sin embargo, el nombre de la ruta absoluta siempre es /A/ D/F/10Además, el nombre de la ruta relativa para acceder a 3 es ../../B/3 .

También puede representar el nombre del directorio actual usando la notación punto (.) La notación punto (.) se usa comúnmente cuando se ejecutan programas que leen el nombre del directorio actual.

¿Con qué comando puede determinar en qué directorio se encuentra actualmente? ¿Existe alguna forma de ingresar a su directorio personal sin necesidad de escribir todo el path completo? ¿Podría utilizar la misma idea para acceder a otros directorios? ¿Cómo? Explique con un ejemplo.

Con pwd.

$HOME

Comando	Función
`cd`	vuelve a su directorio de login
`cd ~`	vuelve también a su directorio de login
`cd /`	le lleva al directorio raíz del sistema completo
`cd /root`	le lleva al directorio principal del root, o superusuario, cuenta creada en la instalacion; debe ser el usuario root para accesar este directorio.
`cd /home`	lo lleva a su directorio principal, donde los directorios login de usuario son almacenados
`cd ..`	le traslada a un directorio superior
`cd ~`*`otheruser`*	le lleva al directorio login del usuario *otheruser*, si *otheruser* le ha dado permiso
`cd /dir1/subdirfoo`	sin tener en cuenta en que directorio esta,

Comando	Función
	este recorrido absoluto le llevara directamente a `subdirfoo`, un subdirectorio de `dir1`
`cd ../../dir3/dir2`	este recorrido relativo lo llevara dos directorios mas arriba, luego a `dir3`, luego al directorio `dir2`.

- cd: cambia de directorio
- mkdir: crea un directorio
- rmdir: elimina un directorio
- mount: monta un dispositivo
- umount: desmonta un dispositivo
- du: muestra lo que ocupa y el tamaño total de los directorios dentro del directorio donde me encuentro
- df: se usa para chequear el espacio en el disco. Mostrará el almacenamiento
- disponible y utilizado de los sistemas de archivos en tu máquina.
- ln: crea enlaces a archivos y crea un fichero que apunta a otro
- ls: lista los archivos y directorios dentro del entorno de trabajo
- pwd: se utiliza para imprimir el nombre del directorio actual.
- cp: El comando cp se emplea para hacer copias de archivos y directorios en nuestro sistema operativo.
- mv: es usado para mover o renombrar archivos o directorios del sistema de archivos.

PROCESOS

Es un programa en ejecución. Un proceso es una entidad "viva" /se modifica/es dinámico, a diferencia de los programas que son estáticos

PID (Process ID): Es un identificador numérico único para cada proceso.

PPID (Process Parent ID): El identificador del proceso padre.

Todos los procesos tienen estos atributos, además de estos (que son los más importantes, pero no todos) usuario (UID), grupo (GID), prioridad.

con ps -ejH puedo ver el PPID

- Mirar todos los procesos que se están ejecutando en el sistema: ps -ef
- Mirar todos los procesos que está ejecutando un usuario: ps -fu usuario
- Filtrar algunas de las columnas de ps: ps -eopid,tt,user,fname,tmout,f,wchan

ps -flU usuario1

F S UID PID PPID C PRI NI ADDR SZ WCHAN STIME TTY TIME CMD

4 S usuario1 5083 5082 0 80 0 - 22179 wait 11:45 pts/1 00:00:00 -bash

0 R usuario1 5118 5083 0 80 0 - 22134 - 11:45 pts/1 00:00:00 ps -f

#

El significado de cada una de las columnas es:

UID → Propietario del proceso.

PID → ID del proceso.

PPID → ID del proceso padre.

C → Cantidad de recursos de CPU que el proceso ha utilizado recientemente para que el kernel (núcleo del sistema operativo) establezca la prioridad apropiada a cada proceso.

PRI → Prioridad del proceso.

NI → Valor nice. Un valor positivo indica menor tiempo de CPU.

STIME → Hora de comienzo del proceso.

TTY → Terminal asociado al proceso.

TIME → Tiempo de CPU asociado al proceso.

CMD → Comando ejecutado

Para ver los procesos en sistemas Linux, contamos con el comando ‘ ps ’, que listará (de múltiples formas según las opciones que le pasemos) todos los procesos que se encuentran corriendo en nuestro equipo.

ps aux (muestra todos los procesos del sistema)

ps axjf (que mostrará un árbol jerárquico con la ruta del programa al que pertenece el proceso)

Aquí un ejemplo de filtrado sobre ps para obtener únicamente los procesos pertenecientes a bash: ps aux | grep bash

El estado background se refiere a que, al ser activado desde una terminal, el proceso se ejecuta de manera independiente a la terminal sin "amarrarla" durante el tiempo de proceso.

El estado foreground en cambio, "amarra" la terminal al proceso dejándola sin capacidad de correr otra tarea en esa terminal mientras el proceso se ejecuta.

Cuando un proceso está en ejecución sin que sea mostrado en la terminal se dice que se está ejecutando en el **background** (segundo plano). Si se muestra la ejecución del comando dentro de la terminal se dice que está en el **foreground** (primer plano).

¿Cómo puedo hacer para ejecutar un proceso en Background? ¿Como puedo hacer para pasar un proceso de background a foreground y viceversa?

Con CTRL + Z es enviar el proceso a segundo plano y queda detenido. Para saber los procesos que están en segundo plano se utiliza el comando jobs.

Para traerlo de background primero se utiliza jobs para saber el número de proceso, y posteriormente fg % seguido del número de proceso (ID).

Los comandos **fg** y **bg** permiten traer un proceso a primer plano o enviarlo a segundo plano respectivamente. Su uso es tan sencillo, como ejecutar la instrucción seguido de % y el número del proceso que obtenemos de jobs.

- traer el proceso a primer plano utilizando fg %2
- devolver el proceso a segundo plano con el atajo de teclado Ctrl+Z
- poner el proceso en marcha una vez que está en segundo plano con bg %2.

Pipe (|)

El "|" nos permite comunicar dos procesos por medio de un pipe desde la shell

El pipe conecta stdout (salida estandar) del primer comando con la stdin (entrada estandar) del segundo.

- Por ejemplo:
    - $ ls | more
        - Se ejecuta el comando ls y la salida del mismo, es enviada como entrada del comanda more

Se pueden anidar tantos pipes como se deseen

-Grep busca una palabra o patrón en un lugar, y devuelve todas las coincidencias.

Redirecciones

En GNU/Linux, al final todo es tratado como si fuera un fichero y como tal, tenemos descriptores de fichero para aquellos puntos donde queramos acceder. Hay unos descriptores de fichero por defecto:

- Entrada estándar (normalmente el teclado).
- Salida estándar (normalmente la consola).
- Salida de error.

Tipo redirección

- Al utilizar redirecciones mediante > (destructiva):
    - Si el archivo de destino no existe, se lo crea.
    - Si el archivo existe, se lo trunca y se escribe el nuevo contenido
- Al utilizar redirecciones mediante >> (no destructiva):
    - Si el archivo de destino no existe, se lo crea
    - Si el archivo existe, se agrega la información al final

Usando el comando cat puedes crear un archivo rápidamente y agregarle texto. Para hacer eso, usa el operador “>” para redirigir el texto en el archivo.

cat > filename.txt

Si hay algún error, lo guarda en fichero (podría salir un error si no tuviéramos permiso de lectura en el directorio).

En el caso del operador < se redirige la entrada estándar, es decir, el contenido del fichero que especificáramos, se pararía como parámetro al comando.

Si quisiéramos redirigir todas las salidas a la vez hacia un mismo fichero, podríamos utilizar >&. Además, con el carácter & podemos redirigir salidas de un tipo hacia otras, por ejemplo, si quisiéramos redirigir la salida de error hacia la salida estándar podríamos indicarlo con: 2>&1. Es importante tener en cuenta que el orden de las redirecciones es significativo: se ejecutarán de izquierda a derecha.

Símbolo	Descripción
>	Redirecciona **stdout** hacía un archivo. Lo crea si no existe, si existe lo sobreescribe.
	`ls -l > lista.txt` (La salida del comando se envía a un archivo en vez de la terminal.)
>>	Redirecciona **stdout** hacía un archivo. Lo crea si no existe, si existe concatena la salida al final de este.
	`ps -ef >> procesos.txt` (Concatena al archivo procesos.txt la salida del comando.)
<	Redirecciona **stdin** desde un archivo. El contenido de un archivo es la entrada o input del comando.
	`mail user < texto.txt` (El cuerpo del correo a enviar proviene desde un archivo, en vez del teclado).)
2> 2>>	Redirecciona **stderr** hacía un archivo. Crea (>) o concatena (>>) la salida de errores a un archivo. (ver ejemplos)
1>&2	Redirecciona **stdout** hacía donde **stderr** apunte. (ver ejemplos)
2>&1	Redirecciona **stderr** hacía donde **stdout** apunte. (ver ejemplos)
OTROS REDIRECCIONAMIENTOS QUE NO UTILIZAN FDs	
<<	Conocido como **HERE-DOCUMENT** o **HereDoc** (ver ejemplos)
<<<	Conocido como **HERE-STRING** (ver ejemplos)
|	El símbolo | (pipe) es un tipo de redireccionamiento ya que la salida (**stdout**) de un comando es la entrada (**stdin**) de otro.
	`ls /etc | grep services` (La salida del comando a la izqueirda de | se convierte en la entrada del comando a la derecha.)
tee	El comando `tee` redirecciona la salida (**stdout**) a ambos, un archivo y a la terminal.
	`ps -ef | tee procesos.txt` (La salida de `ps` se muestra en la terminal y al mismo tiempo se redirecciona al archivo *procesos.txt*. Con la opción *-a* (`tee -a`) concatena al archivo.)

kill es un comando unix/linux que permite enviar señales a uno o varios procesos del sistema a través de la shell (bash, ksh, etc). Las señales más utilizadas suelen ser la de matar un proceso (9 ó SIGKILL), pararlo (TERM) o reiniciarlo (1 ó HUP) pero hay muchas más que pueden ser útiles en ocasiones. El listado completo de señales disponibles puede visualizarse ejecutando kill -l

ps: Muestra información de los procesos activos.

ps [OPCIONES]

Por defecto muestra:

- **PID** : Id del Proceso.
- **TTY** : Terminal.
- **TIME** : Tiempo de ejecución.
- **CMD** : Comando.

Opciones

- **a** : Muestra todos los procesos asociados a un TTY.
- **-e** / **-A** : Muestra todos los procesos.
- **x** : Muestra los no asociados.
- -**f** : Muestra el formato largo:
    - **UID** : Usuario que lo ejecutó.
    - **PPID** : Id del proceso padre.
    - **C** : Uso del procesador.
    - **STIME** : Inicio de ejecución.
- **u** : Orientado al usuario:
    - **USER**
    - **% CPU** : uso de procesador.
    - **% MEM** : uso de memoria.
    - **VSZ** : Memoria virtual.
    - **RSS** : Memoria física.
    - **STAT** : Estado.
    - **START** : Iniciado.

Significados de los estados STAT :

- **S** : Esperando (Sleep).
- **R** : Ejecutando (Running)
- **D** : Esperando entrada/salida.
- **T** : Pausa.
- **Z** : No responde (Zombie)

Información adicional de los STAT:

- **s** : Proceso padre
- **l** : Proceso con hilos.
- **+** : En primer plano.

pstree: muestra en forma de árbol los procesos en ejecución.

Con el parámetro “-a”, nos muestra la línea de comandos utilizada. Por ejemplo, si el comando utiliza la ruta a un fichero de configuración.

Por defecto se inhabilita la visualización en el árbol de los nombres repetidos. Para evitar esto, podemos utilizar el parámetro “-c”

Si nos interesa podemos ver el árbol de un proceso específico, de la siguiente manera:

pstree -H [PID]

top: muestra en tiempo real información del sistema. (Por defecto cada 3 segundos)

$ top [-d SEGUNDOS] [-u USUARIO]

htop: con htop podremos obtener un resultado más amigable, no esta instalado por defecto se ha de instalar mediante $ apt-get install htop ... una vez instalado ...

$ htop

killall: mata todos los procesos con un nombre concreto o de un usuario o ambos.

$ killall [-u USUARIO] [-SENAL] [NOMBRE_PROCESO]

Ejemplos de uso:

Matar todos los procesos con el nombre de "sleep"

$ killall sleep

Matar todos los procesos del usuario "Pepito"

$ killall -u Pepito

Matar todos los procesos del usuario "Pepito" que se llamen "sleep"

$ killall -u Pepito sleep

nice: ejecuta un comando con una prioridad distintas ala de por defecto. Solo los usuarios root pueden establecer prioridades urgentes (negativos).

$ nice -n NUMERO_PRIORIDAD COMANDO

renice: cambia la prioridad de un proceso ejecutándose. No se puede aumentar la urgencia.

$ renice -n NUMERO_PRIORIDAD COMANDO

$ nice -10 named (esto bajaría la prioridad de named en -10 unidades (si estaba en -10, pasara a -20) MENOS GENTIL = MAS PRIORIDAD

¿A qué hace referencia el concepto de empaquetar archivos en GNU/Linux?

- Empaquetar: Es agrupar en un solo fichero varios ficheros y/o directorios.
- Comprimir: Es reducir el tamaño de un fichero a través del uso de un algoritmo de compresión.

Empaquetar

En linux contamos con el comando tar, que nos permite realizar el proceso de empaquetación, su sintaxis es la siguiente:

tar [opciones] [nombre_fichero_tar] [directorio_origen]

Las opciones más utilizadas son (la versión con un guion es la corta y con dos guiones la larga, pero hacen lo mismo):

-c --create: Crea un nuevo archivo.

-x --extract: Extrae ficheros de un archivo.

-v --verbose: Lista detalladamente los ficheros procesados.

-f \$&fichero\$&: Empaqueta o desempaqueta en o hacia un fichero.

-t --list: Lista los contenidos de un archivo.

Algunos ejemplos de uso son los siguientes:

- Crear un archivo tar llamado edteam.tar con los archivos del directorio cursos.

  ```
  tar -cf edteam.tar cursos
  ```

- Crear un archivo tar llamado edteam.tar con los archivos del directorio cursos mostrando el detalle de los ficheros procesados.

  ```
  tar -cvf edteam.tar cursos
  ```

- Ver el contenido del archivo edteam.tar

  ```
  tar -tf edteam.tar
  ```

Comprimir

Los comandos gzip y gunzip permiten comprimir y descomprimir ficheros respectivamente, su sintaxis básica es:

```
gzip [archivo_a_comprimir]
```

```
gunzip [archivo_a_descomprimir]
```

gzip edteam.tar

gunzip edteam.tar.gz

Podemos realizar el proceso de empaquetación y compresión a través de una sola instrucción, agregando la opción -z al comando tar.

Veamos algunos ejemplos:

- Crear un archivo empaquetado y comprimido llamado edteam.tar.gz con los archivos del directorio cursos mostrando el detalle de los ficheros procesados.

  ```
  tar -cvzf edteam.tar.gz cursos
  ```

- Desempaqueta y descomprime el archivo edteam.tar.gz mostrando el detalle de los ficheros procesados.
  `tar -xvzf edteam.tar.gz`

Comprimir con zip

Además de gzip y unzip, podemos comprimir y descomprimir a través de zip y unzip respectivamente, este formato de compresión es el más utilizado en sistemas operativos Windows.

Su sintaxis básica es la siguiente:

```
zip -r [nombre_fichero_zip] [directorio_a_comprimir]

unzip [nombre_fichero_zip]
```

- Crear un archivo comprimido llamado edteam.zip con los archivos del directorio cursos:
  `zip -r edteam.zip cursos`
- Descomprimir el archivo edteam.zip
  `unzip edteam.zip`

Empaquetar archivos:	`tar cvf` *`nombre.tar directorio_o_nombres_archivos`*
Desempaquetar archivos:	`tar xvf` *`nombre.tar`*
Comprimir archivos:	`gzip` *`nombre_archivo_a_comprimir`*
Descomprimir archivos:	`gzip -x` *`nombre_archivo_a_descomprimir`*

¿Pueden comprimirse un conjunto de archivos utilizando un único comando?

No, se deben comprimir de a uno

tar : permite empaquetar varios archivos en uno solo, sin comprimirlos. Tambien sirva para desempaquetar.

Ej: tar –vcf nombre_archivo.tar nombre_carpeta_a_empaquetar

Ej: tar –vxf mi_archivo.tar

grep : muestra líneas que concuerdan con un patrón.

gzip : comprime o expande ficheros.

zgrep : busca una expresión regular en ficheros posiblemente comprimidos.

wc : sirve para contar líneas, palabras y caracteres que contiene un archivo.

FIND

find / -(i)name “nombre” → un archivo en particular

ej find / -name my-file → busca archivo llamado my-file

ej find / -iname my-file → busca archivo llamado my-file sin distinguir mayúsculas de minusculas

find / -not -name my-file → busca archivo sin my-file en el nombre

find / -name “*.txt” → características similares

/ a partir del directorio raíz

. a partir de donde estoy parado

~ directorio personal

Búsqueda por tipo:

d – directorio o carpeta

f – archivo normal

l – enlace simbólico

c – dispositivos de caracteres

b – dispositivos de bloque

find / -type d → todos los directorios en mi sistema de archivos

find / -type f -name my-file → esto buscará archivos llamados my-file, excluyendo directorios o enlaces.

find / -name “LEAME”

EJEMPLOS COMANDOS IMPORTANTES

cp ∗ /home −R → intenta copiar todo lo que esta en el directorio donde se esta parado a home. El -R indica que se haga lo mismo con los archivos y subcarpetas internos

rm ∗ → borra todos los archivos del directorio en el que se esta parado

Desempaqueta y descomprime en otro directorio.

tar -xvf misLogs.tar **-C** /home/agusnfr/dir1

tar -xzvf misLogs.tar.gz **-C** /home/agusnfr/dir2

PRACTICA 3

Shell Scripting

Un Shell Script es un programa que está creado con instrucciones que son ejecutadas por un Shell (CLI o intérprete de comandos) de Unix o Linux. El código no es compilado ni precompilado, se va ejecutando línea por línea efectuando lo que cada instrucción le indica. Necesita un programa que entienda los comandos y estructuras que contiene y esto se suele poner en la primera línea del programa. Por ejemplo #!/bin/bash significa que le pasaremos al BASH las lineas del fichero de Script. Dado que el BASH es el intérprete de comandos más famoso de Linux, los Script que se crean para este entorno también se pueden llamar Bash Script.

Sirve para:

- Automatización de tareas.
- Aplicaciones interactivas.
- Aplicaciones con interfaz gráfica (con el comando zenity, por ejemplo).

echo: imprime el texto.

read: lee una línea desde entrada estándar.

- Para crear una variable:

```
NOMBRE="pepe" # SIN espacios alrededor del =
```

- Para accederla se usa $:

```
echo $NOMBRE
```

- Para evitar ambiguedades se pueden usar llaves:

```
# Esto no accede a $NOMBRE
echo $NOMBREesto_no_es_parte_de_la_variable
# Esto sí
echo ${NOMBRE}esto_no_es_parte_de_la_variable
```

Parametrización

- Los *scripts* pueden recibir argumentos en su invocación.
- Para accederlos, se utilizan variables especiales:
  - `$0` contiene la invocación al script.
  - `$1`, `$2`, `$3`, ... contienen cada uno de los argumentos.
  - `$#` contiene la cantidad de argumentos recibidos.
  - `$*` contiene la lista de todos los argumentos.
  - `$?` contiene en todo momento el valor de retorno del último comando ejecutado.

$HOME es el directorio de trabajo por defecto del usuario.

Comando exit

Para terminar un *script* usualmente se utiliza la función `exit`:

- Causa la terminación de un *script*
- Puede devolver cualquier valor entre 0 y 255:
  - El valor 0 indica que el script se ejecutó de forma exitosa
  - Un valor distinto indica un código de error
  - Se puede consultar el *exit status* imprimiendo la variable `$?`

EXPR

Este comando devuelve en su salida estándar el resultado de las expresiones aritméticas pasadas como argumentos. Su sintaxis es expr expresión.

Todos los elementos de la expresión deben ir separados por al menos un espacio, y ciertos operadores aritméticos llevan como prefijo una barra invertida para evitar toda confusión con los caracteres especiales del shell.

Operaciones básicas: suma, resta, multiplicación, división y módulo en números enteros.

Otras: evaluación de expresiones regulares, operaciones de cadena como subcadena, longitud de cadenas, etc.

Los operadores aritméticos son:

+: suma

-: resta

*: multiplicación

/: división entera

%: resto de la división entera o módulo

\(y \): paréntesis

Se utiliza generalmente una sustitución de comandos para asignar el resultado del comando expr a una variable. Se obtiene por ejemplo:

```
[agusnfr]$ expr 2 + 3
5
[agusnfr]$ expr 2 - 3
-1
[agusnfr]$ expr 2 + 3 \* 4
14
```

```
[agusnfr]$ expr \( 2 + 3 \) \* 4
20
[agusnfr]$ resultado=$(expr 9 / 2)
[agusnfr]$ echo $resultado
4
[agusnfr]$ expr $resultado % 3
1
```

expr length "PEPE"

https://linuxhint.com/linux-expr-command/ mas info ahí

Comando test

El comando test permite efectuar una serie de pruebas sobre los archivos, las cadenas de caracteres, los valores aritméticos y el entorno de usuario.

Este comando tiene un código de retorno igual a cero cuando el test es positivo, y diferente de cero en caso contrario; esto permite utilizarlo en encadenamientos de comandos con ejecución condicional (&& y ||) o en las estructuras de control que veremos más adelante.

El comando test posee dos sintaxis: test expresión y [expresión], donde "expresión" representa el test que se debe efectuar.

Los espacios detrás del corchete de apertura y antes del corchete de cierre son obligatorios en la sintaxis [expresión]. En general, todos los elementos de sintaxis del comando test deben ir separados por al menos un espacio.

También se pueden utilizar dobles corchetes [[, para realizar comparaciones. Los dobles corchetes resultan ser una mejora respecto a los simples. Así, las diferencias entre uno y otro son las siguientes,

1. No hay que utilizar las comillas con las variables, los dobles corchetes trabajan perfectamente con los espacios. Así [-f "$file"] es equivalente a [[-f $file]].
2. Con [[se puede utilizar los operadores || y &&, así como<y>` para las comparaciones de cadena.
3. Se puede utilizar el operador =~ para expresiones regulares, como por ejemplo [[$respuesta =~ ^s(i)?$]]
4. También se puede utilizar comodines como por ejemplo en la expresión [[abc = a*]]

Por compatibilidad se utilizan simples.

COMPARANDO CADENAS

[[[

>	>
<	<
=	=
!=	!=

COMPARANDO ENTEROS

[[	[	
-gt	-gt	mayor
-lt	-lt	menor
-ge	-ge	mayor o igual
-le	-le	menor o igual
-eq	-eq	igual
-ne	-ne	distinto

OPERADORES BOOLEANOS

[[	[
&&	-a
||	-o

- ⇒ -d fichero => cierto si fichero existe y es un directorio
- ⇒ -e fichero => cierto si fichero existe, independientemente del tipo que sea
- ⇒ -f fichero => cierto si fichero existe y es un fichero normal
- ⇒ -r fichero => cierto si fichero existe y se puede leer
- ⇒ -s fichero => cierto si fichero existe y tiene tamaño mayor que cero
- ⇒ -w fichero => cierto si fichero existe y es se puede escribir sobre él
- ⇒ -x fichero => cierto si fichero existe y es ejecutable

chequear -e Existe

chequear -d Directorio

chequear -f Archivo

chequear -L Link

chequear -r Lectura

chequear -w Escritura

chequear -x Ejecución

```bash
#!/bin/bash

# Para validar la cantidad de parámetros de un script o una función, podemos
# utilizar el comando `test` y algunos de sus test flags:

if [ $# -ne 3 ]; then
  echo El script no recibió 3 parámetros
  # Lo más común en este punto sería retornar un exit code > 0 (error)
  # exit 1
fi

if [ $# -lt 2 ]; then
  echo El script recibió menos de 2 parámetros
  # exit 2
fi

if [ $# -gt 0 ] && [ -z $1 ]; then
  echo El script recibió parámetros, pero el primero está en blanco
  # exit 3
fi
```

```bash
#!/bin/bash

# Como en otros lenguajes de programación, bash dispone de estructuras de
# control que permiten tomar decisiones condicionales, realizar operaciones
# repetitivas, etc.
#
# Antes de introducir las estructuras de control, es necesario comprender
# cómo funcionan los valores booleanos (verdadero o falso) en bash: el cero es
# considerado verdadero, mientras que cualquier otro valor es considerado falso.
# ¿Se ve la relación con el exit status de los scripts?
#
# Existen dos comandos que simbolizan el verdadero y falso: `true` y `false`
# respectivamente. ¿Cómo funcionan? Utilizan su exit status para representar
# la veracidad o falsedad de su valor:

true
echo "El comando true retornó el exit status => $?"

false
echo "El comando false retornó el exit status => $?"

# Existe un comando en bash que permite evaluar condiciones lógicas: el comando
# `test`. Este comando recibe diferentes parámetros que representan una
# expresión lógica, la evalúa y retorna un valor booleano. Un alias para el
# test son los corchetes ([ ]) -- tener en cuenta que siempre llevan espacios
# alrededor! Para conocer en detalle las posibilidades que brinda este comando,
# se puede ver su documentación mediante el siguiente comando:

info test
```

IF

if [condition]

then

    block

elif

    block

else

fi

CASE

Selección:

```
case $variable in
        "valor 1")
                block
        ;;
        "valor 2")
                block
        ;;
        *)
                block
        ;;
esac
```

SELECT

```
select variable in opcion1 opcion2 opcion3
do
        # en $variable está el valor elegido
         block
done
```

Menú de opciones:

Ejemplo:

```
select action in New Exit
do
case $action in
        "New")
                echo "
                    Selected
                     option
                    is NEW"
        ;;
        "Exit")
                exit 0
        ;;
        esac
done
```

Imprime:

y espera el número de opción por teclado

FOR

- C-*style*:

```
for ((i=0; i < 10; i++))
do
        block
done
```

- Con lista de valores (`foreach`):

```
for i in value1 value2 value3 valueN;
do
        block
done
```

WHILE

while

```
while [ condition ] #Mientras se cumpla la condición
do
        block
done
```

UNTIL

until

```
until [ condition ] #Mientras NO se cumpla la condición
do
        block
done
```

break [n] corta la ejecución de n niveles de loops.

continue [n] salta a la siguiente iteración del enésimo loop que contiene esta instrucción.

Las **variables** son de tipo string o arreglos.

El shell es un lenguaje débilmente tipado (las variables en un idioma están fuertemente tipadas si tienen tipos explícitos; las variables sin tipos débiles son lenguajes débilmente tipados) y son diferentes de C. La definición de variables en la programación de shell no requiere un tipo, y no existe un concepto de tipo.

ARREGLOS

- Creación:

```
arreglo_a=() # Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) # Inicializado
```

- Asignación de un valor en una posición concreta:

```
arreglo_b[2]=spam
```

- Acceso a un valor del arreglo (En este caso las llaves no son opcionales):

```
echo ${arreglo_b[2]}
copia=${arreglo_b[2]}
```

- Acceso a todos los valores del arreglo:

```
echo ${arreglo[@]} # o bien ${arreglo[*]}
```

- Tamaño del arreglo:

```
${#arreglo[@]} # o bien ${#arreglo[*]}
```

- Borrado de un elemento (reduce el tamaño del arreglo pero no elimina la posición, solamente la deja vacía):

```
unset arreglo[2]
```

- Los índices en los arreglos comienzan en 0

FUNCIONES

Las funciones permiten modularizar el comportamiento de los *scripts*.

- Se pueden declarar de 2 formas:
  - `function nombre { block }`
  - `nombre() { block }`
- Con la sentencia `return` se retorna un valor entre 0 y 255
- El valor de retorno se puede evaluar mediante la variable `$?`
- Reciben argumentos en las variables `$1`, `$2`, etc.

- Las variables no inicializadas son reemplazadas por un valor nulo o 0, según el contexto de evaluación.
- Por defecto las variables son globales.
- Una variable local a una función se define con `local`

```
test() {
        local variable
}
```

- Las variables de entorno son heredadas por los procesos hijos.
- Para exponer una variable global a los procesos hijos se usa el comando `export`:

```
export VARIABLE_GLOBAL="Mi var global"
comando
# comando verá entre sus variables de
# entorno a VARIABLE_GLOBAL
```

- No hacen falta, a menos que:
  - el *string* tenga espacios.
  - que sea una variable cuyo contenido pueda tener espacios.
  - son importantes en las condiciones de los `if`, `while`, etc...
- Tipos de comillas
  - **"Comillas dobles":**

```
var='variables'
echo "Permiten usar $var"
echo "Y resultados de comandos $(ls)"
```

  - **'Comillas simples':**

```
echo 'No permiten usar $var'
echo 'Tampoco resultados de comandos $(ls)'
```

```
Un ejemplo:
variable="un texto de varias palabras"
variable_2=UnaSolaPalabra
echo "Podemos leer $variable"
echo 'No podemos leer $variable'
variable_3="Asi concateno $variable_2 a otro string"
```

- Permite utilizar la salida de un comando como si fuese una cadena de texto normal.
- Permite guardarlo en variables o utilizarlos directamente.
- Se la puede utilizar de dos formas, cada una con distintas reglas:

```
$(comando_valido)
`comando_valido`
```

Nota: La primer forma resulta más clara y posee reglas de anidamiento de comandos más sencillas.

- Ejemplo:

```
arch="$(ls)" # == arch="`ls`" == arch=`ls`
mis_archivos="$(ls /home/$(whoami))"
```

ANOTACIONES IMPORTANTES

^

Principio de cadena (línea), grep '^h' notas muestra líneas de notas que empieza por h

LET

Let evalua cada argumento como una expresión aritmética. Las operaciones se evaluan como enteros, mientras que una división por cero arrojará un error.

De nuevo, las cosas se ven mas claras con unos ejemplos,

let z=25; echo $z Devuelve 25

let z++; echo $z Devuelve 26

let z=z+10; echo $z. Devuelve 36

DOBLES PARÉNTESIS

Otra opción para realizar operaciones matemáticas, es el uso de los dobles paréntesis. Al fin y al cabo, los dobles paréntesis se comportan como let que acabas de ver. Aunque tienen la ventaja de que un doble paréntesis puede incluir a otro doble paréntesis. De esta manera, los ejemplos que te he indicado para let se pueden hacer exactamente igual para el caso de los dobles paréntesis,

((z=25)); echo $z Devuelve 25

((z++)); echo $z Devuelve 26

((z=z+10)); echo $z. Devuelve 36

Pero además, si quieres devolver el resultado de una operación simplemente tienes que preceder el doble paréntesis con $, es decir, puedes simplificar los ejemplos anteriores de la siguiente manera,

echo $((z=25)); Devuelve 25

echo $((z++)); Devuelve 26

echo $((z=z+10));. Devuelve 36

Esto va a permitir, como he adelantado anteriormente, incluir una operación matemática dentro de otra. Por ejemplo,

`echo $((25 + $((5+5))))

o incluso echo $((25 + $((diez=5+5))));echo $diez

EQUIVALENCIAS

cat archivo | wc -c # equivale a: wc -c archivo | cut -d' ' -f1

find -name archivo # equivale a: find . -name archivo

grep ac archivo # equivale a: cat archivo | grep ac

echo hola | cat > salida # equivale a: echo hola > salida

echo $((expresión)) → Equivale a: expr expresión

"cadena", $(./archivo) → Equivale a: `./archivo`

cadena=`./archivo` → Guarda lo que imprime "archivo" al ejecutarse adentro de la variable "cadena"

<u>TEST EJEMPLO</u>

test 1 -eq 2 && echo “true” || echo “false”

si es igual, se imprime true, si no, false

(test -f archivo && grep menta archivo && echo Z) || echo Q

Si "archivo" existe y contiene el string "menta" imprime "Z", sino imprime "Q"

Sin los paréntesis el resultado sería el mismo

<u>Contar</u>

wc -l <fichero> número de líneas

wc -c <fichero> número de bytes

wc -m <fichero> imprime el número de caracteres

wc -L <fichero> imprime la longitud de la línea más larga

wc -w <fichero> imprime el número de palabras

Cortar

cut opciones [ARCHIVO]

Donde las opciones son:

-b rango/s

Selecciona los bytes indicados en el o los rangos. También puede indicarse con el parámetro --bytes=rango/s.

-c rango/s

Selecciona los caracteres indicados en el o los rangos. También puede indicarse con el parámetro --characters=rango/s.

-f rango/s

Selecciona los campos (columnas) indicadas en el o los rangos, que se encuentran delimitadas por el carácter Tabulador. También puede indicarse con el parámetro --fields=rango/s. En caso de que una línea no posea el delimitador, esta se muestra en su totalidad, salvo que se utilice el parámetro -s.

-d caracter_delimitador

Especifica el carácter delimitador de los campos, en el caso de que se utilice el parámetro -f. Es posible indicarlo usando --delimiter=caracter_delimitador. En caso de que no se use dicho parámetro el tabulador actúa como delimitador por defecto.

-s

Indica que las líneas que no posean el delimitador (o separador) no sean mostradas.

--complement

Establece que los rangos indicados en alguna de las opciones -b, -c o -f no son deseados, por lo tanto se muestra el resto que se encuentra excluido.

--output-delimiter=cadena_de_texto

Indica que los campos, al ser mostrados en pantalla, sean separados por la cadena de texto indicada.

En cada invocación de cut, solo es posible elegir una de las maneras de segmentar las líneas de un texto. Esto limita al usuario a optar por un único parámetro entre los siguientes: -b, -c o -f.

Ejemplo

Quedarme con la primer columna de un texto separado por: desde entrada estándar

cut -d: -f1

TR

tr [OPTION] SET1 [SET2]

Ejemplo: tr '[:lower:][:upper:]' '[:upper:][:lower:]' (cambia minúsculas por mayúsculas y mayúsculas por minuscular)

tr no es más que una abreviación de translate. Esto ya nos puede dar la pista que las principales utilidades de tr son las siguientes:

1. Cambiar caracteres, palabras y frases de mayúscula a minúscula o viceversa.
2. Buscar palabras y reemplazarlas por otras.
3. Borrar caracteres, palabras o frases.
4. Eliminar caracteres repetidos de forma innecesaria.
5. Cambiar letras o palabras de mayúscula a minúscula.
6. Eliminar la totalidad de caracteres que son ilegibles de un texto.
7. Añadir una nueva línea cada vez que encontremos un delimitador que podemos definir.
8. Eliminar la totalidad de caracteres de puntuación de una frase.
9. Etc.

Si queremos borrar todas las vocales mayúsculas y minúsculas de una frase y que sean reemplazadas por un espacio en blanco lo haremos mediante la opción tr -d.

-c, -C, --complement

Usa el complemento del CONJUNTO1. Esto significa que define al CONJUNTO1 como todos los caracteres que no se encuentran en la definición dada por el Usuario. Este parámetro es útil para indicar caracteres que no queremos que se vean afectados.

-d, --delete

Borra los caracteres de definidos en CONJUNTO1.

-s, --squeeze-repeats

Elimina la secuencia continua de caracteres repetidos, definidos en el CONJUNTO1.

-t, --truncate-set1

Trunca la longitud del CONJUNTO1 según la longitud del CONJUNTO2. Esto hace que si el primer conjunto es más largo que el segundo, solo sean tenidos en cuenta n primeros caracteres, siendo n la longitud de CONJUNTO2.

joan@gk55:~$ echo "Esto es un ejemplo de tr para el blog geekland" | tr -cd 'aeiou'

oeueeoeaaeoeea

Nota: El significado de -cd 'aeiou' vendría a ser borra todo aquello que no sean vocales a, e, i, o, u.

LISTA COMANDOS

File Commands

ls - directory listing
ls -al - formatted listing with hidden files
cd *dir* - change directory to *dir*
cd - change to home
pwd - show current directory
mkdir *dir* - create a directory *dir*
rm *file* - delete *file*
rm -r *dir* - delete directory *dir*
rm -f *file* - force remove *file*
rm -rf *dir* - force remove directory *dir* *
cp *file1 file2* - copy *file1* to *file2*
cp -r *dir1 dir2* - copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* - rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* - create symbolic link *link* to *file*
touch *file* - create or update *file*
cat > *file* - places standard input into *file*
more *file* - output the contents of *file*
head *file* - output the first 10 lines of *file*
tail *file* - output the last 10 lines of *file*
tail -f *file* - output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps - display your currently active processes
top - display all running processes
kill *pid* - kill process id *pid*
killall *proc* - kill all processes named *proc* *

System Info

date - show the current date and time
cal - show this month's calendar
uptime - show current uptime
w - display who is online
whoami - who you are logged in as
finger *user* - display information about *user*
uname -a - show kernel information
cat /proc/cpuinfo - cpu information
cat /proc/meminfo - memory information
man *command* - show the manual for *command*
df - show disk usage
du - show directory space usage
free - show memory and swap usage
whereis *app* - show possible locations of *app*
which *app* - show which *app* will be run by default

Compression

tar cf *file.tar files* - create a tar named *file.tar* containing *files*
tar xf *file.tar* - extract the files from *file.tar*
tar czf *file.tar.gz files* - create a tar with Gzip compression
tar xzf *file.tar.gz* - extract a tar using Gzip
tar cjf *file.tar.bz2* - create a tar with Bzip2 compression
tar xjf *file.tar.bz2* - extract a tar using Bzip2
gzip *file* - compresses *file* and renames it to *file.gz*
gzip -d *file.gz* - decompresses *file.gz* back to *file*

Referencia de Comandos en Linux

Traducción: Alex Barrios (alexbariv@gmail.com) / Basado en la publicación original de Jacob en FOSSWire

Manejo de Archivos

ls – listar directorio
ls -al – listado con formato y mostrando ocultos
cd *dir* – cambiar a directorio "dir"
cd – cambiar a directorio home
pwd – muestra el directorio actual
mkdir *dir* – crear el directorio "dir"
rm *archivo* – borrar archivo
rm -r *dir* – borrar directorio "dir"
rm -f *archivo* – forzar el borrar archivo
rm -rf *dir* – forzar borrar directorio de forma recursiva
cp *archivo1 archivo2* – copiar *archivo1 a archivo2*
cp -r *dir1 dir2* – copiar *dir1 a dir2*; Creando *dir2 si no exíste*
mv *archivo1 archivo2* – renombrar o mover *archivo1* a *archivo2*. Si el *archivo2* es un directorio, mueve *archivo1* al contenido de ese directorio
ln -s *archivo link* – crea un enlace simbólico de link a *archivo*
touch archivo – crea o actualiza un archivo
cat > archivo – coloca la salida estándar en *archivo*
more archivo – muestra el contenido de un archivo
head archivo – muestra las primeras 10 líneas de un archivo
tail archivo – muestra las últimas 10 líneas de un archivo
tail -f *file* – muestra las últimas 10 líneas de en tiempo real

Gestión de procesos

ps – muestra los procesos activos actualmente
top – muestra todos los procesos
kill *pid* – mata un proceso indicando el *pid*
killall *proc* – mata un proceso llamado *proc*
bg – lista los procesos detenidos o trabajando en fondo; puede resumir procesos
fg – trae el proceso más reciente al frente
fg *n* – trae N procesos al frente

Permisología

chmod *octal archivo* – cambia los permisos del *archivo* con *octal*, que pueden ser identificados por separado el usuario, grupo o mundo añadiendo:
* 4 – leer (r)
* 2 – escribir (w)
* 1 – ejecutar (x)
Ejemplos:
chmod 777 – leer, escribir, y ejecutar para todos
chmod 755 – rwx para el dueño, rx para el grupo y mundo
Para más opciones, observa: **man chmod**.

Uso de SSH

ssh *usuario@host* – conecta a *usuario* en *host*
ssh -p *puerto user@host* – conecta a el *host* en el *puerto* con el usuario *user*
ssh-copy-id *user@host* – añade tu llave a el *host* para activar el inicio de sesión sin clave

Búsquedas

grep *patrón archivos* – busca en los archivos por el *patrón*
grep -r patrón *dir* – busca recursivamente el *patrón* en los directorios.
comando | grep *patrón* – busca por el *patrón en la salida del comando*
locate *archivo* – encuentra todas las instancias del archivo

Información del Sistema

date – muestra la hora y fecha actual
cal – muestra el calendario del mes
uptime – muestra el tiempo en ejecución del sistema
w – muestra quién está conectado
whoami – muestra como quién está conectado
uname -a – muestra información del kernel
cat /proc/cpuinfo – información del cpu
cat /proc/meminfo – información de la memoria
man *comando* – muestra el manual para el *comando*
df – muestra el uso de disco
du – muestra el uso de disco del directorio
free – muestra la memoria ram y swap libre
whereis app – muestra las posibles ubicaciones de *app*
which *app* – muestra cual *app* corre por defecto

Compresión

tar cf *file.tar archivos* – crear un archivo tar llamado *file.tar* que contiene *archivos*
tar xf *file.tar* – extrae los contenidos de *file.tar*
tar czf *file.tar.gz files* – crea un tar comprimido con Gzip
tar xzf *file.tar.gz* – extrae un tar que usa Gzip
tar cjf *file.tar.bz2* – crea un tar comprimido con Bzip2
tar xjf *file.tar.bz2* – extrae un tar que usa Bzip2
gzip *file* – comprime *file y lo renombra a file.gz*
gzip -d *file.gz* – descomprime *file.gz* de vuelta a *file*

Redes

Ping *host* – ejecuta ping a *host y muestra los resultados*
whois *dominio* – obtiene la info whois de un dominio
dig *dominio* – obtiene la info DNS de un dominio
dig -x *host* – busca el reverso DNS del *host*
wget archivo – descarga un archivo
wget -c archivo – continua una descarga pausada

Instalando Software

Instalando desde las fuentes (normalmente un tar.gz):
./configure
make
make install
Con sistemas de paquetes
dpkg -i *pkg.deb* – instala un paquete (Debian)
rpm -Uvh *pkg.rpm* – instala un paquete (RPM)

Atajos de teclado

Ctrl+C – detiene el comando actual
Ctrl+Z – pausa el comando actual, lo resumes con **fg** al frente o **bg** en el fondo
Ctrl+D – sierra la sesión, similar a **exit**
Ctrl+W – borra una palabra de la linea actual
Ctrl+U – borra toda la línea
Ctrl+R – repite el último comando
exit – sale de la sesión actual

Filtros

cat [*f1*] ... concatena ficheros.
p.76 sin ficheros: lee $\mathcal{EE}$.

cmp *f1 f2* compara ficheros. (T/F) p.92

comm compara conjuntos
`-13` id. sin columnas 1^a ni 3^a

```
let421 let431
       cuatro
dos
    tres
p.90        uno
```

cut `-d`*delim* `-f`*lista* deja campos de *lista*.
`-c`*lista* id. caracteres.
p.75 `-c2,5-7,10-` 2^o, 5^o .. 7^o, 10^o.. fin.

diff *f1 f2* diferencia entre ficheros texto.
`-e` secuencia-ed, lleva de *f1* a *f2*
p.93 `-c`[*n*] dif. con contexto: *n* (3) líneas

egrep como `grep`, $\mathcal{ER}$ modernas p.88,105

fgrep *tira* [*f1*] .. busca líneas con *tira*
`-v` id. sin *tira*
`-f` *fich* [*f1*] .. busca líneas de *fich*
p.88 en *f1* o en $\mathcal{EE}$

grep $\mathcal{ER}$ [*f1*] .. busca líneas con $\mathcal{ER}$.
p.88,105 `-v` id. sin $\mathcal{ER}$.

head [-*n*] *n* (10) primeras líneas. p.74

join *f1 f2* une: claves comunes,
producto cart. restos línea.
`-d`*del* ... id. con delimitador *del*.
`-nf` *nc* clave: campo *nc*, fichero *nf*
`-o` *listaCamposSalida*
p.77 `-o 1.2` campo 2^o fichero 1^o, ..

nl numera líneas no blancas.
p.97 `-ba` id. todas.

patch con *f1* y `diff -c` obtiene *f2* p.96

paste [*f1*] ... concatena líneas.
p.76 `-d`*del* [*f1*] .id. con delimitador *del*.

pr [*f1*] .. prepara para imprimir
`-m` ficheros en columnas paralelas
`-n` cada fichero *n* columnas
`-w`*n* anchura (def. 72)
`-l`*n* longitud (def. 66)
`+`*n* salta $n-1$ pags.
`-h` *cab* pone otro encabezamiento.
`-t` suprime líneas previas,
p.97 finales, y cabecera.

rev [*f1*] ... invierte cada línea p.100

sed [*f1*] ... editor no interactivo: a $\mathcal{SE}$
`-e` *com* instrucciones en línea.
`-f` *fich* instrucciones en el fichero.
p.81 `-n` no copia implícita a $\mathcal{SE}$
dónde - función - parámetros
dónde (nada) todas las líneas
3 línea tercera
6,$-2 líneas sexta .. última - 2
/$\mathcal{ER}$/ líneas contienen $\mathcal{ER}$
/$\mathcal{ER}_1$/,/$\mathcal{ER}_2$/ desde $\mathcal{ER}_1$hasta $\mathcal{ER}_2$
p.83 *dónde*! complemento de *dónde*
func. p escribe
d borra
p.84 r *fich* incluye *fich* a continuación.
s/$\mathcal{ER}_1$/*tira*$_2$/ substituye 1^a $\mathcal{ER}_1$
p.123 por *tira*$_2$
s/.../g id. todas las apariciones

sort [*f1*] ... ordena (ASCII)
`-k 3,5` id. según campos 3..5
`+2 -5 +7` id. s. campos 3,4,5,8.. †
[+n_1] [-n_2] ... (1^{er} campo: 0)
`-t`*delim* delimitador de campos
`-r` orden decreciente
`-n` ordena números
p.86 `-u` quita líneas repetidas

split [*f1*] parte fichero/$\mathcal{EE}$.
cada 1000 líneas.
en `xaa`, `xab`, ...
[-*n*] [*f1*] id. cada *n* líneas.
f1 rz id. en *rz*`aa`, *rz*`ab`, ...
p.75 - *rz* id. entrada estándar ($\mathcal{EE}$).

tac [*f1*] ... invierte orden líneas p.100

tail [-*n*] *n* (10) últimas líneas.
[-n +*p*] desde la línea *p* al final.
[±*n*b] id. bloques. (512 oct)
p.74 [±*n*c] id. caracteres.

tr *sec1 sec2* cambia carac. id. posición.
[-c] complemento *sec1*
[-s] elimina susbs. repetidas.
p.80 [-d] *sec1* borra caracteres en *sec1*.

uniq quita líneas iguales consecutivas.
`-c` id. con recuento.
p.88 `-d` iguales consecutivas: deja 1 línea

wc [*f1*] .. cuenta líneas, palabras y caract.
`-l` sólo líneas
`-w` palabras
p.99 `-c` caracteres.

comandos

aspell `list` [*f*] lista pals. desconocidas
p.205 `check` *f* para corregir *f*

bc calculadora
`scale=`*n* *n* decimales
`obase=`*n* base (salida)
p.204 `ibase=`*n* base (entrada)

btoa convierte: binario → ASCII
p.212 `-a` id. ASCII → binario

cal saca calendario del mes actual
a id. del año *a*
p.203 *m a* id. del mes *m* del año *a*

cd trabajar en *directorio de entrada*
p.61 *dir* id. en *dir*

chgrp *gr f* .. asocia *f* al grupo *gr* p.122

chmod *mod f* .. cambia permisos de *f*
[*quien*] ± *perm*
[u] [g] [o] dueño, grupo, otros
[r] [w] [x] leer, escribir, ejecut/inodo
p.114 `744` `rwxr--r--`

chown *us f* .. da *f* a *us* p.121

cp *f1 f2* copia ficheros, *f1* a *f2*
p.37,63 *f1 ... dir* id. *f1* a *dir/f1*
p.46 `-p` mantiene fechas, dueño, ..

date [+*fm*] escribe fecha (formato *fm*)

df espacio libre en discos/particiones p.223

du [*dir*] disco ocupado, cada directorio
`-a` id. cada fichero
p.222 `-s` id. en resumen (*dir*)

echo escribe parámetros en $\mathcal{SE}$ p.12

exit acaba sesión (`sh`) p.3

file tipo/contenido de objeto/fichero p.12

find *dir cond1* .. busca en *dir* qué objetos
p.218 cumplen las condicion(es)
`-name` *nm* su nombre es *nm*
`-mtime` [±]*n* modificado hace (±)*n* días
`-atime` [±]*n* accedido hace (±)*n* días
`-size` [±]*n* tamaño (±)*n* bloques
`-newer` *f* más reciente que *f*
`-type c` fichero
`-type d` directorio
`-inum` *n* i-nodo *n*
`-print` imprime (T)
`-exec` *com* .. `\;` ejecuta *com*

kill *pid* termina el proceso *pid* (señ.15)
`-9` ya!
p.127 *-señal* envía esa señal

last [n] últimas (*n*) sesiones
p.26 *us* id. del usuario *us*

less [*f1*].. presenta pantalla a pantalla p.16

ln *f1 f2* *f2* : otro nombre de *f1*
p.37,63 *f1 ... dir* *dir/f1*: otro nombre de *f1*
p.42 `-s` *f1 f2* *f2* apunta a *f1*

logout acaba sesión (`sh`) p.4

lpq presenta la cola de impresión p.19

lpr [*f1*] pone en cola para imprimir p.19

lprm [*n*] quita un trabajo (*n*)
p.19 de la cola de impresión

ls [*dir*] lista nombres (ficheros,
dirs, etc.) en *dir*
`-l` id. con más información
p.10 `-a` y nombres empiezan por '.'
p.38 `-i` con número de *i-nodo*
p.58 `-F` indica tipo de objeto
`-d` sólo información del directorio
p.62 `-R` aplica recursivamente a subdir.
p.10 `--full-time` con fecha completa

make [*ob*] actualiza el objetivo *ob*
`-n` sólo indica las acciones
p.167 `-s` actualiza en silencio

man *com* manual del comando *com* p.21

mkdir *dir1 dir2* crea directorio(s) p.62

more [*f1*].. presenta pantalla a pantalla p.15

mutt lee correo
`-f` *fch* lee correo guardado en *fch*

Comando	Descripción
cd directorio	Ir al directorio.
ls	Muestra los contenidos del directorio.
pwd	Devuelve el directorio actual.
rm archivo	Borra archivo.
rm -r carpeta	Borra carpeta.
mv archivo destino	Mueve archivo a destino.
cp archivo destino	Mueve archivo a destino.
mkdir carpeta	Crea una carpeta
nano archivo	Edita archivo en la consola.
find directorio [opciones] [tests] [acciones]	Busca en directorio archivos que cumplan [tests] y hace lo que indiquen [acciones]
man comando	Muestra la ayuda e instrucciones de comando
comando --help	Ayuda breve de comando.
killall nombredeproceso	Mata a todos los procesos llamados nombredeproceso
comando1 && comando2	Ejecuta comando1 y si no da error, ejecuta después comando2
comando1; comando2	Ejecuta comando1 y después comando2.
comando1 | comando2	Manda la salida de comando1 a comando2.
comando1 | grep palabra	Muestra sólo las líneas de salida de comando1 que contienen palabra.
comando > archivo	Guarda la salida de comando en archivo
comando | tee archivo	Guarda la salida de comando en archivo y la muestra por la terminal a la vez.
ifup -a / ifdown -a	Habilita/deshabilita todas las conexiones de red (requiere root)
mount disco puntodemontaje	Monta disco en puntodemontaje (requiere root).
umount disco/puntodemontaje	Desmonta disco o puntodemontaje (requiere root).
sudo comando	Ejecuta comando con permisos de administrador.
su / sudo -i	Entra en modo root permanente.
Control+Alt+FX	Salta a la terminal virtual X.
fuser archivo	Muestra qué procesos están usando archivo.
fuser -k archivo	Muestra qué procesos están usando archivo y los mata.
$(comando)	Se sustituye al ejecutar por la salida de comando.

LS

- Ls para listar el contenido del directorio de trabajo actúa
- ls [ruta de directorio aquí] para listar el contenido de otro directorio
- ls / para listar el contenido del directorio raíz:
- ls .. comando para listar el contenido del directorio padre un nivel más arriba. Usa ls ../.. para contenidos dos niveles arriba:
- ls ~ para listar el contenido en el directorio personal del usuario:
- ls -d */ para listar solo los directorios:
- ls * para listar el contenido del directorio con sus subdirectorios:
- ls -R para listar todos los archivos y directorios con sus subdirectorios correspondientes hasta el último archivo:
- ls -s (la s es minúscula) para listar archivos o directorios con sus tamaños:
- ls -l para listar el contenido del directorio en un formato de tabla con columnas incluidas:
    - permisos de contenido
    - número de enlaces al contenido
    - propietario del contenido
    - propietario del grupo del contenido
    - tamaño del contenido en bytes
    - fecha / hora de la última modificación del contenido
    - nombre de archivo o directorio

- ls -lh para listar los archivos o directorios en el mismo formato de tabla anterior, pero con otra columna que represente el tamaño de cada archivo/directorio:
- ls -a para listar archivos o directorios, incluidos archivos o directorios ocultos. En Linux, cualquier cosa que comience con un . se considera un archivo oculto:
- ls -t para listar archivos o directorios y ordenar por fecha de última modificación en orden descendente (de mayor a menor).
    - -r para invertir el orden de clasificación de la siguiente manera: ls -tr:
- ls -S (la S es mayúscula) para listar archivos o directorios y ordenar por tamaño en orden descendente (de mayor a menor).
    - -r para invertir el orden de clasificación de la siguiente manera: ls -Sr: