

A thick black L-shaped frame surrounds the text. It starts at the top left, goes right, then down, then right again at the bottom right.

PROGRAMACIÓN DE SISTEMAS

Unidad 1 – Programación en el Shell

PRÓLOGO



¿Qué es la Programación de Sistemas?

- Es la programación a bajo nivel, para la creación de software del sistema.
- Ejemplos de software del sistema:
 - *Compiladores*
 - *Bases de datos*
 - *Servidores web, etc.*
- Aplicaciones GUI viven en un nivel más alto.

¿Por qué Programación de Sistemas?

- El software de sistema constituyen la infraestructura sobre las cuales corren las aplicaciones de más alto nivel.
- Por ejemplo, si usamos Java, alguien tuvo que implementar la JVM.
- Antes de usar bases de datos, alguien tuvo que implementar el motor de base de datos.

¿Por qué Programación de Sistemas?

- En este curso, haremos programación de sistemas en Linux.
- Nuestro lenguaje de implementación será **C**.
- Los programadores de sistema hacen uso de las **llamadas al sistema**.

1.1 INTRODUCCIÓN A LA ARQUITECTURA DE LINUX



Introducción a la Arquitectura de Linux

- ¿Que es UNIX?

UNIX fue originalmente concebido como una "mesa de trabajo" para desarrollo de aplicaciones dentro de Bell Labs en los 70s. Terminó convirtiéndose en SO.

- Creado por:



Ken Thompson



Dennis Ritchie (R.I.P.)

Introducción a la Arquitectura de Linux

- UNIX introdujo conceptos importantes:

- *Sistemas de archivos jerárquico*
- *Procesos*
- *Archivos de dispositivos*
- *Intérprete de línea de comandos*
- *Redirección de entrada/salida*
- *Time-sharing*
- *Multiprogramación*

Introducción a la Arquitectura de Linux



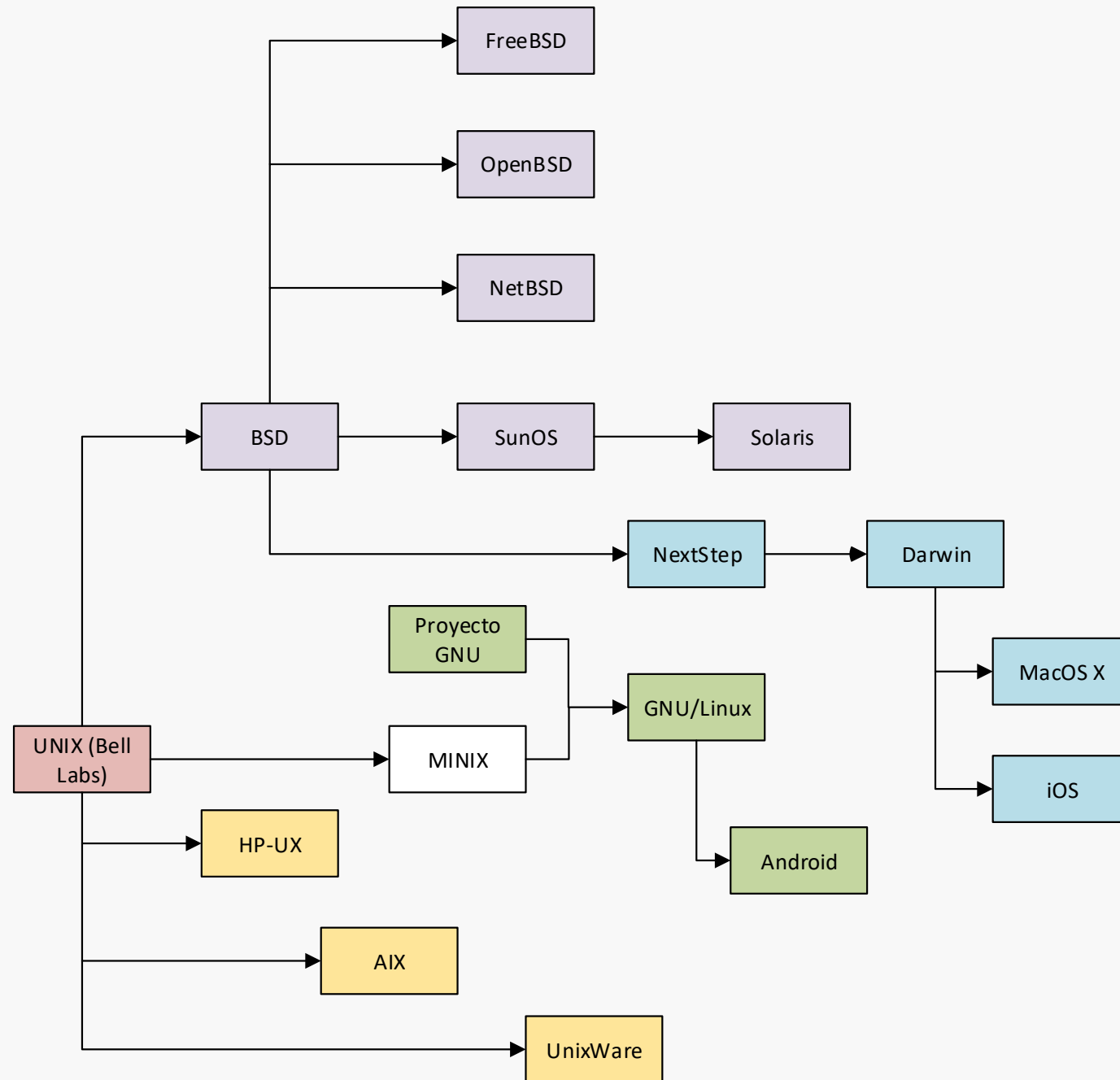
PDP-7



PDP-11

Introducción a la Arquitectura de Linux

- UNIX fue escrito ensamblador y en C (también creado por Thompson y Ritchie)
 - *Por lo tanto existe una **relación estrecha** entre C y UNIX.*
- UNIX está estandarizado en la Single UNIX Specification v4 (POSIX:2008).



Introducción a la Arquitectura de Linux

- ¿Qué es Linux?
 - *Linux es un sistema operativo similar a UNIX (**Unix-like**)*
 - *El componente principal de Linux es el núcleo (kernel), desarrollado por Linus Torvalds en 1991.*
- **Kernel:** el principal componente del sistema operativo , que maneja los aspectos más básicos del hardware y da servicios a las aplicaciones.

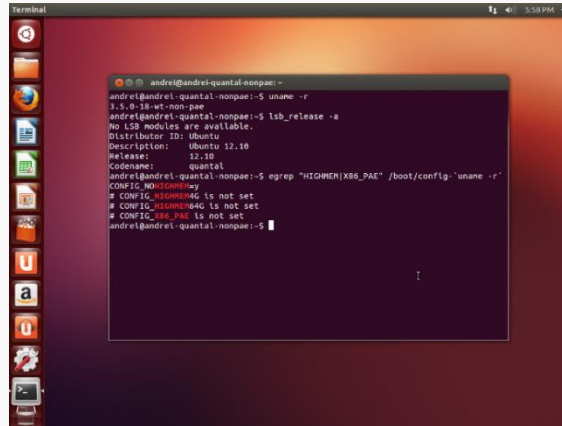
Introducción a la Arquitectura de Linux

- Un Sistema *Unix-like* implementa la interface POSIX (Portable Operating System Interface).
- POSIX define interfaces de programación del sistema operativo tales como:
 - *Interfaces al Sistema*
 - *Definiciones base*
 - *Comandos y utilidades*
 - <http://pubs.opengroup.org/onlinepubs/9699919799/>

¿Dónde encontramos a Linux?



Smartphones



PCs



Centros de computo/servidores



Sistemas embebidos



Aplicaciones en tiempo real

Introducción a la Arquitectura de Linux

■ Estándarización

- **POSIX:** *Portable Operating System Interface, creada por la IEEE.*
- *El estándar 1003.1 especifica una interfaz (API), no implementación*
- *Define los servicios que el SO debe proveer para ser considerado compatible con POSIX*
- *¡Windows no es POSIX compliant!*
- *Define los archivos cabeceras (.h) necesarios.*

Introducción a la Arquitectura de Linux

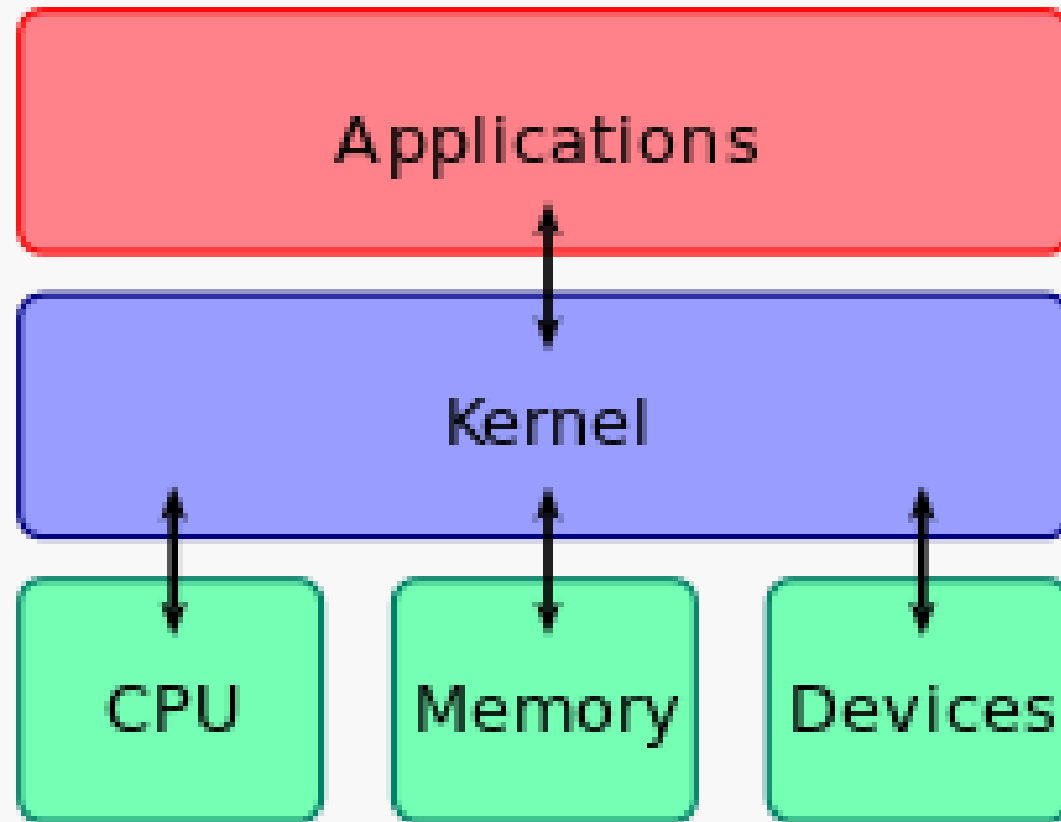
■ Conceptos Generales

- *UNIX/Linux es un sistema operativo. Como tal, cumple dos funciones básicas:*

- Proveer una **abstracción** del hardware.
- Proveer servicios a los programas que corren sobre él.

- **Kernel:** parte fundamental del sistema operativo que controla los recursos de hardware.

- **Llamadas al sistema:** interface de programación para acceder a los servicios ofrecidos por el *kernel*.



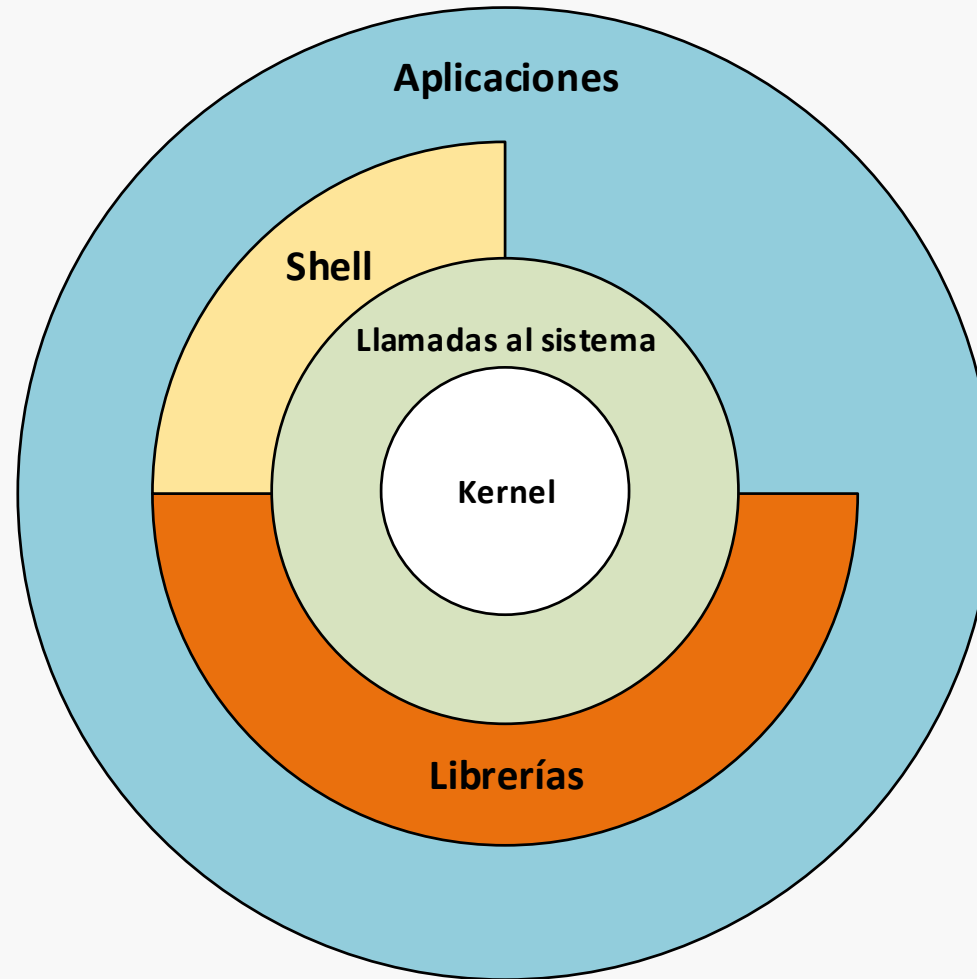
Introducción a la Arquitectura de Linux

■ Abstracción

Nivel establecido de complejidad de interacción entre sistemas; los **detalles** de la implementación se manejan en un nivel inferior.

Hardware	Abstracción
CPU	Hilos
Memoria	Proceso
Disco	Archivos
Red	Comunicacion (sockets)
Monitor	Gráficos, ventanas, salida
Teclado	Entrada
Mouse	Puntero

Introducción a la Arquitectura de Linux



Introducción a la Arquitectura de Linux

- Al conectarnos a un sistema Linux, debemos proveer usuario y contraseña.
 - *El sistema verificara el archivo `/etc/passwd` (en la actualidad se usan un archivo encriptado).*
 - *Este archivo contiene líneas como:*

sar:x: 205: 105: Stephen Rago:/home/sar:/bin/ksh

- *Siete columans: usuario, contraseña encriptada, ID de usuario (205), ID de grupo (105), campo comentario, directorio home, programa shell*

Introducción a la Arquitectura de Linux

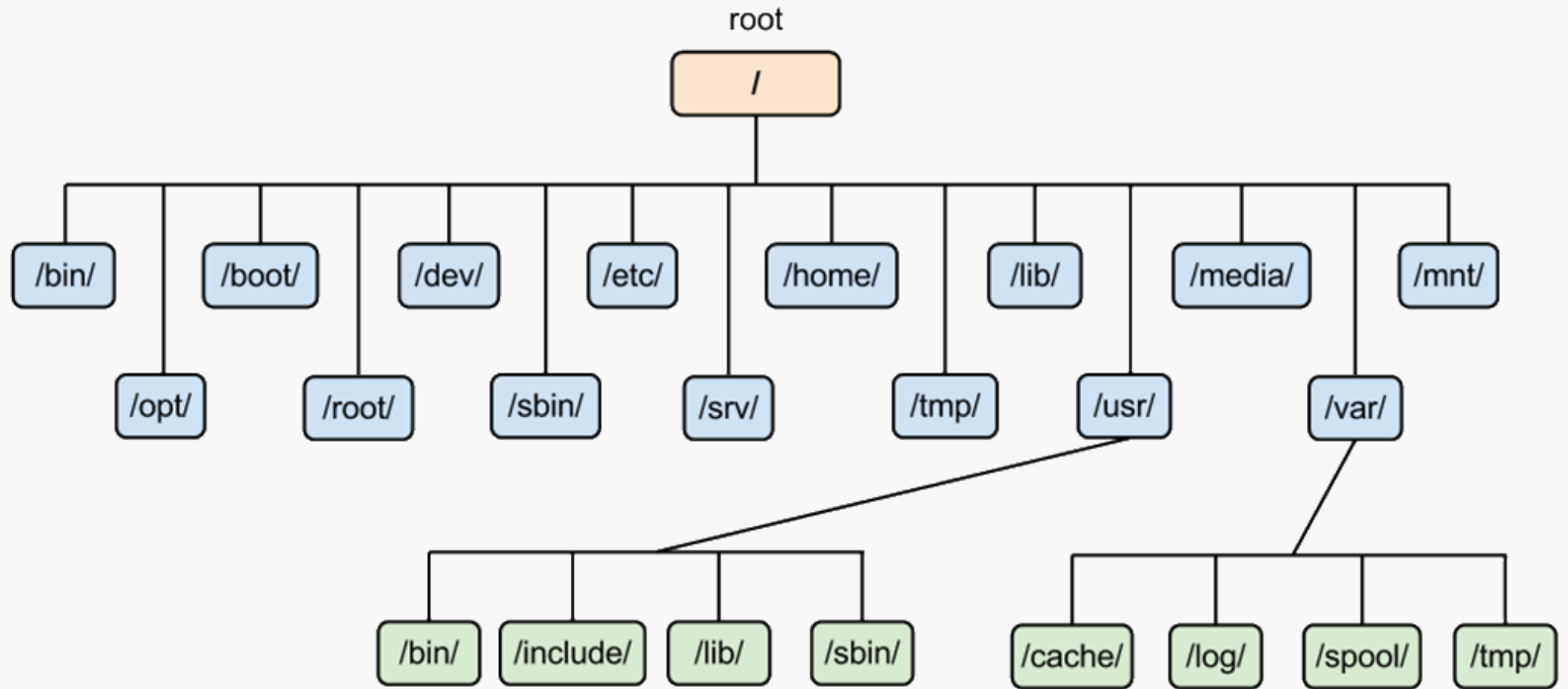
- **Shell:** intérprete de línea de comandos que lee la entrada del usuario y ejecuta comandos.

Shell	Ruta	FreeBSD	Linux 3.2	Mac OS X	Solaris 10
Bourne Shell	/bin/sh	si	si	copia de bash	si
Bourne-again Shell	/bin/bash	opcional	si	si	si
C Shell	/bin/csh	link a tcsh	opcional	link a tcsh	si
Korn Shell	/bin/ksh	opcional	opcional	si	si
TENEX C Shell	/bin/tcsh	si	opcional	si	si

Introducción a la Arquitectura de Linux

Sistema de Archivos

- Sistema de archivos de UNIX/Linux es jerárquico (árbol)
 - *Todo empieza en el directorio root (/)*
- **Directorio:** archivo que contiene entradas de directorio
- **Entrada de directorio:** nombre de archivo + número de i-nodo



Introducción a la Arquitectura de Linux

- Sistema de archivos virtual (VFS): Es una abstracción del sistema de archivos actual (AFS)
 - *Provee una interfaz entre el AFS y las aplicaciones*
- Sistema de archivos actual (AFS): Implementación real del sistema de archivos.
 - *Ext4*
 - *NFS*
 - *LogFS*
 - *ReiserFS*
 - *ZFS*

Introducción a la Arquitectura de Linux

- **Nombres de archivo:** el caracter '/' ni el character nulo pueden ser parte del nombre del archivo.
 - *¿Por qué?*
- Dos archivos son creados al crear un directorio:
 - *El archivo . → apunta al propio directorio*
 - *El archivo .. → apuntar al directorio padre*
- ¿A dónde apunta . y .. en el directorio root (/)?

Introducción a la Arquitectura de Linux

- **Ruta de archivos:** secuencia de nombre de archivos separados por slashes

`/home/root/ntpdate`

- *Ruta absoluta: relativa al directorio root (/)*
- *Ruta relativa: ruta relativa al **directorio de trabajo**.*

- **Directorio de trabajo:** directorio desde el cual se resuelven las rutas

Introducción a la Arquitectura de Linux

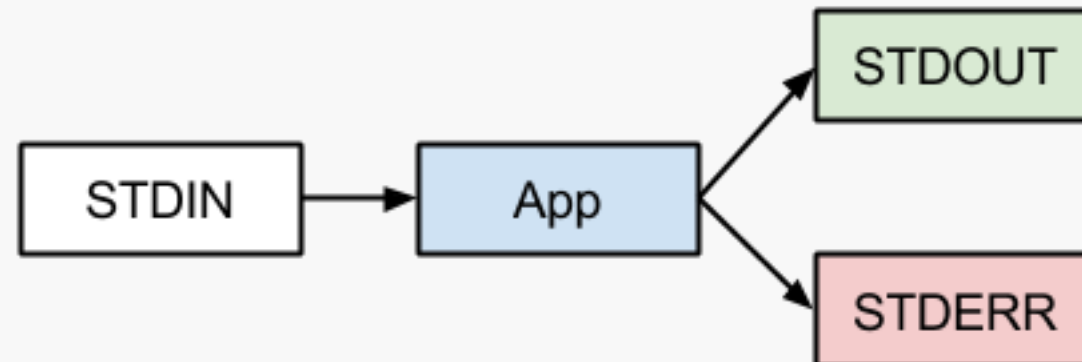
Entrada/Salida

- Para leer/escribir cosas en Linux, lo hacemos a través de descriptores de archivo.
- **Descriptores de archivos:** Numero entero no negativo para referirse a archivo abierto.

Introducción a la Arquitectura de Linux

Entrada, salida y error estándar.

- *stdin* → descriptor 0 (usualmente el teclado)
- *stdout* → descriptor 1 (usualmente la terminal/shell)
- *stderr* → descriptor 2 (usualmente la terminal/shell)



Introducción a la Arquitectura de Linux

Permisos

- Cada usuario tiene un ID (user ID)
- Usuarios se agrupan en grupos con ID (group ID)
- El *user ID* y *group ID* nos ayudará a saber que acciones son permitidas en que archivos/Directorios.

Introducción a la Arquitectura de Linux

- Todo archivo/directorio tiene 9 bits que nos dicen los permisos:

rw - ***w*** ***x*** ***rw*** ***x***

- **Rojo** → Permisos de dueño
- **Verde** → Permisos del grupo del dueño
- **Azul** → Permisos de otros grupo
- **r** – permiso lectura, **w** – permiso escritura, **x** – permiso de ejecución.

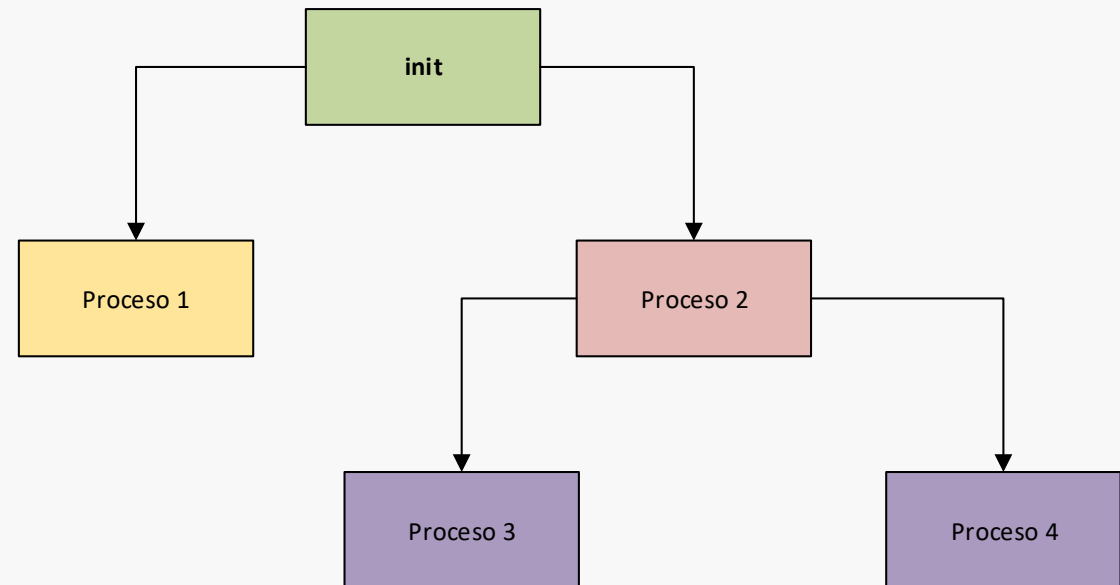
Introducción a la Arquitectura de Linux

Programas y Procesos

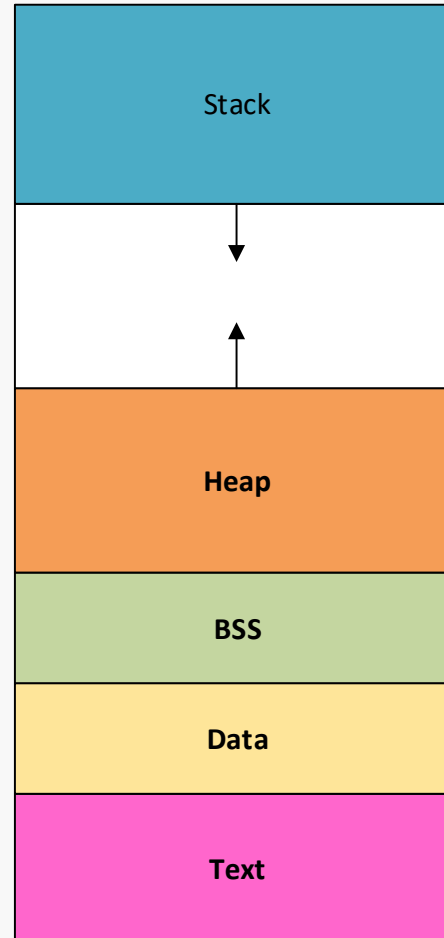
- **Programa:** archivo ejecutable almacenado en disco.
- **Proceso:** instancia de un **programa** en ejecución.
- Todo proceso tiene un identificador único (PID) garantizado por el SO.

Introducción a la Arquitectura de Linux

- Los procesos en Linux siguen una jerarquía de árbol. Todo proceso "nace" de otro proceso.
- El proceso inicial es el proceso *init*



- Espacio de direcciones (address space) de un proceso (memoria virtual).



Introducción a la Arquitectura de Linux

Hilos

- Abstracción del CPU.
- Usualmente un hilo por proceso
- Con múltiples hilos, podemos hacer mejor uso del CPU (conurrencia).
- Hilos son implementados por la librería *pthread*

Introducción a la Arquitectura de Linux

Manejo de Errores

- **errno:** variable global donde se guarda el último error.
- En Linux cuando ocurre un error:
 - *Función retorna -1.*
 - *Se establece la variable `errno`*
 - *La variable `errno` y los códigos de error están definidos en `errno.h`*

Introducción a la Arquitectura de Linux

■ Tipos de error

○ *Fatales*

- No hay recuperación de estos errores.

○ *No-fatales*

- Usualmente por falta de recursos. Posible recuperación es intentar más tarde.

■ Dos funciones básicas para manejo de errores

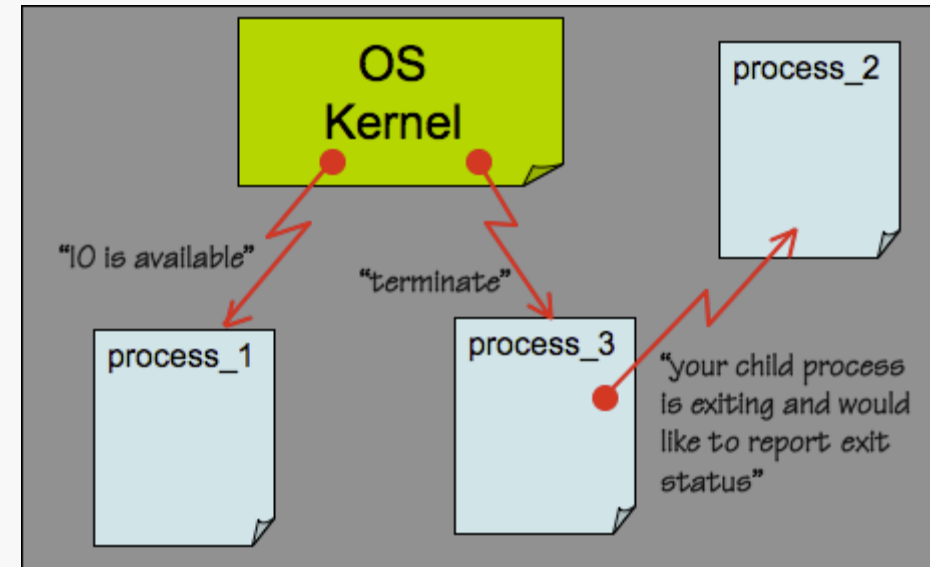
- `char *strerror(int errnum)`

- `void perror(const char *msg)`

Introducción a la Arquitectura de Linux

Señales

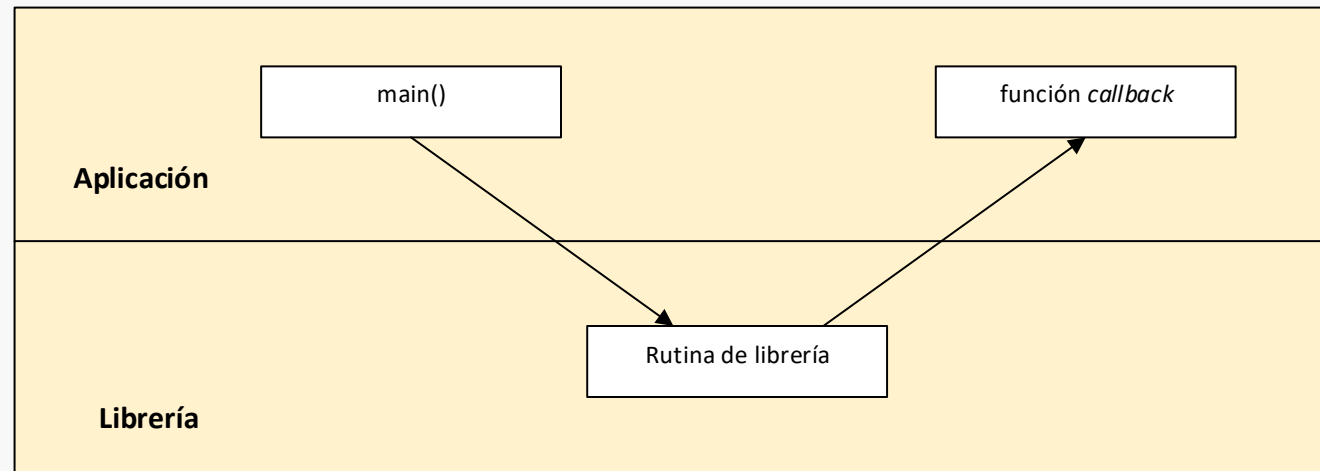
- Mecanismo para notificar a un proceso que algo ha ocurrido
- El manejo de señales en Linux/UNIX hace uso de funciones *callback*



Introducción a la Arquitectura de Linux

¿Qué es una función *callback*?

- Es una función que pasada como argumento a otra función, que se espera que la llame.



Introducción a la Arquitectura de Linux

Tiempo en Linux

- Dos tipos de tiempo:
 - **Tiempo calendario:** valor que contiene el número de segundos desde 00:00:00 Enero , 1970.
 - **Tiempo de proceso (tiempo de CPU):** mide tiempo de ejecución de un proceso
 - Medido el clock ticks
 - 50, 60 o 100 ticks por segundos

1.2 LLAMADAS AL SISTEMA



Llamadas al Sistema y Librerías

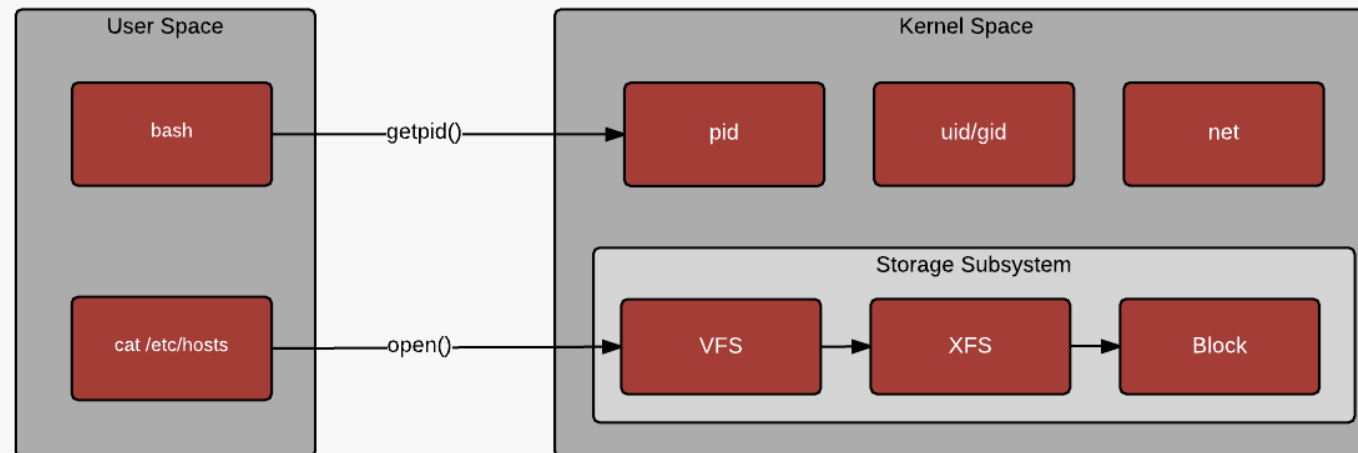
- **Llamada al sistema:** interfaz de programación para acceder a los servicios ofrecidos por el *kernel*.
 - El numero de llamadas al sistema disponibles varia por SO.
- Definidas en lenguaje C.
 - En Linux, la técnica es tener una **función** en la librería standard C con el mismo nombre de la llamada al sistema.
- Para nosotros, las llamadas al sistema son básicamente funciones en C.

Llamadas al Sistema

- SO mantiene un **contexto** para cada proceso corriendo.
- Llamadas al sistema son caras en términos de tiempo, ya que requieren dos cambios de contexto.
 - *Uno para ir de espacio de usuario al kernel*
 - *Otro para ir del kernel a espacio de usuario.*
- **Cambio de contexto:** Es el cambio entre modo KERNEL y de USUARIO.

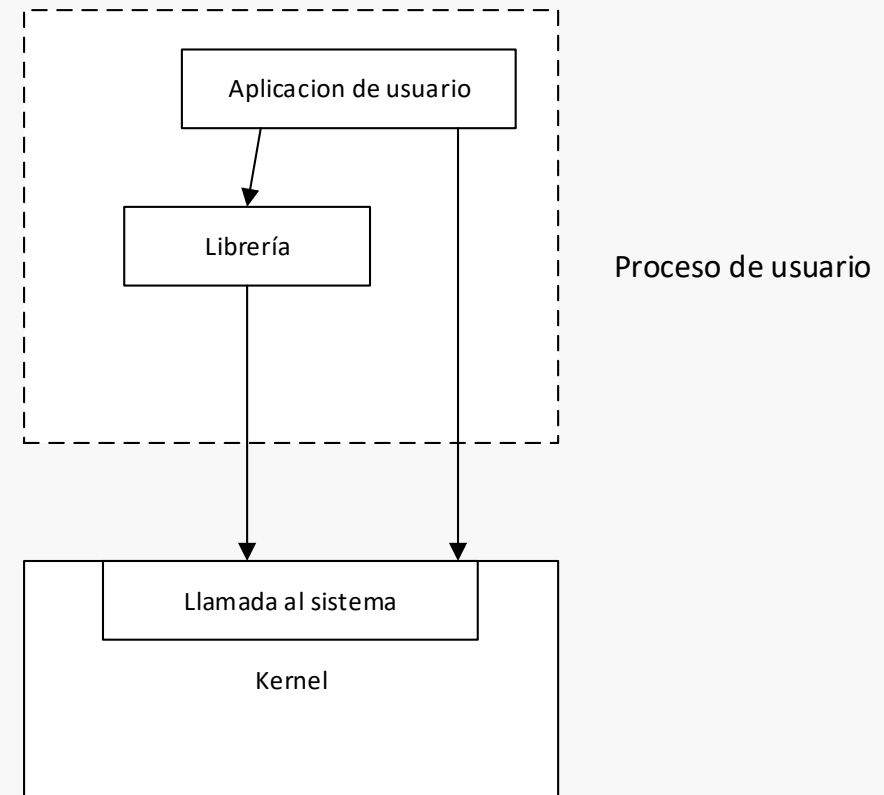
Llamadas al sistema

- CPU tiene dos modos de ejecución:
 - Modo privilegiado → cuando estamos en espacio de kernel.
 - Modo de usuario → cuando estamos en espacio de usuario.



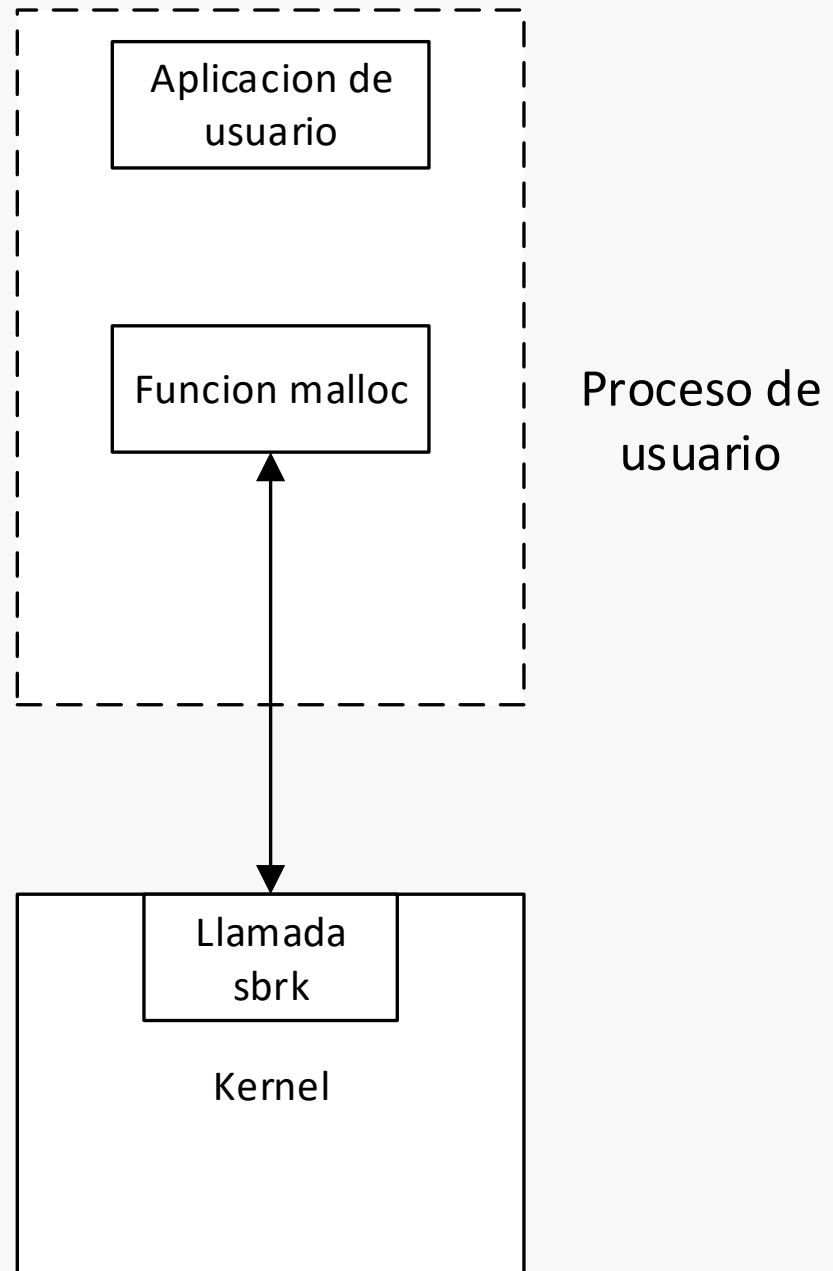
Llamadas al Sistema

- Aplicaciones usar a las llamadas al sistema:
 - *Directamente.*
 - *A través de librerías:*



Llamadas al sistema

- Ejemplo: `write()`
- Aplicación puede llamar a librería o llamada al sistema.
- Rutinas de librería pueden hacer llamadas al sistema.
- Llamadas al sistema suelen tener interfaz mínima, mientras que rutinas de librerías proveen funcionalidad más elaborada.



1.3 SHELL SCRIPTS



Shell Scripts

¿Por qué scripting?

Los scripts permiten automatizar las tareas del administrador. Estos pueden ser desde scripts sencillos hasta scripts altamente complejos.

¿En qué podemos escribir los scripts?

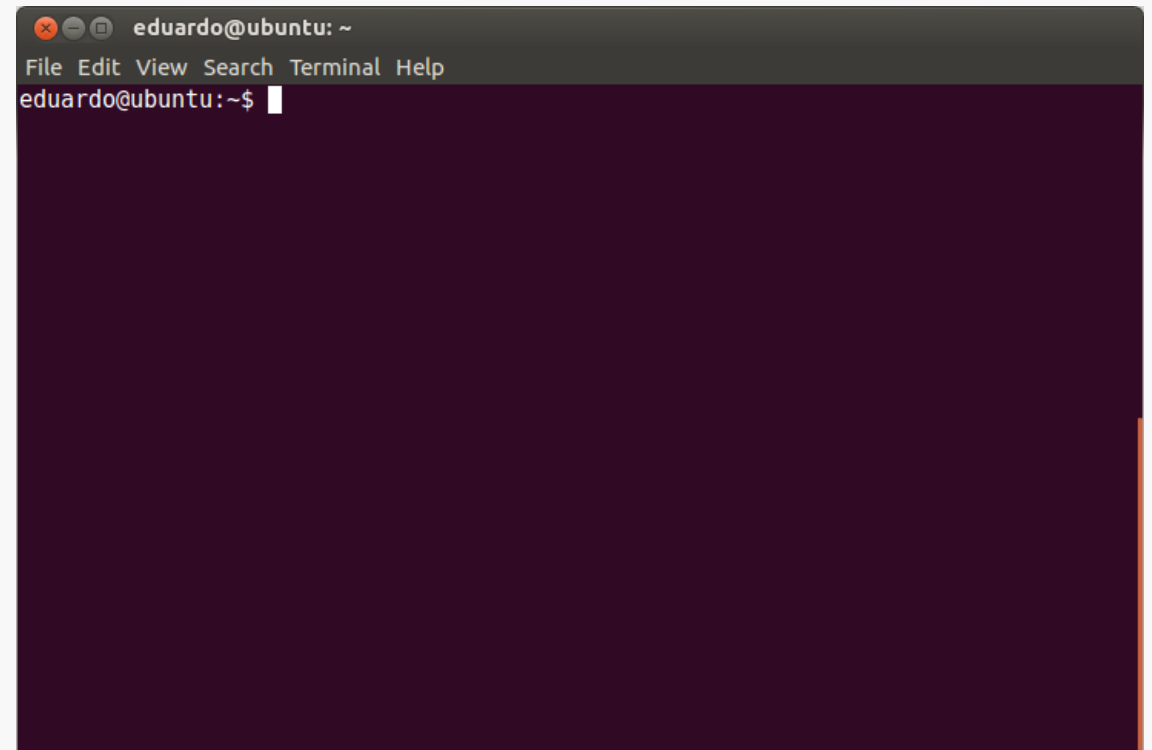
Python, Perl o el shell. En este curso, usaremos shell scripts. Para scripts complejos, ya debemos usar lenguajes como Perl o Python.

Ahora, conozcamos a nuestro mejor amigo, **el shell**.

Shell Scripts

El Shell

Es la principal herramienta del administrador. Es un intérprete de línea de comandos:



Shell Scripts

- El shell es quien interpreta los comandos que el usuario escribe, o que están en un script.
 - **sh (Bourne Shell):** *el shell original de UNIX.*
 - **bash (Bourne Again Shell):** *el shell estándar de Linux. Flexible e intuitive.*
 - **csh (C Shell):** *Shell con sintaxis similar a C.*
 - **tcsh (TENEX C Shell):** *superset del C shell*
 - **ksh (Korn Shell):** *superset del Bourne shell.*

Shell Scripts

- Los comandos que podemos ejecutar son de cuatro tipos:

1. **Programas:** *scripts o programas compilados.*
2. **Comando propio del shell:** *por ejemplo el comando cd.*
3. **Función del shell:** *scripts miniatura del shell.*
4. **Alias:** *comandos definidos por uno mismo, usando otros comandos.*

Shell Scripts

¿Cómo uso el shell?

Empezemos por lo más fácil. En el shell podemos escribir cualquier cosa:

abc

En este caso, el comando no existe. Escribamos algo que **exista...**

Shell Scripts

Por ejemplo, usemos el comando para mostrar el contenido de una carpeta (**ls**):

```
$ ls
```

Luego que tipeamos el comando, este se **ejecuta**, y los resultados son mostrados en pantalla.

Shell Scripts

Argumentos

- Los comandos también pueden recibir **argumentos**. Por ejemplo, mandemos el argumento `-l` (este le dice a `ls` que muestre detalles de los archivos):

```
$ ls -l
```

Shell Scripts

También podemos mandar **múltiples argumentos**. Los podemos mandar por separado, o juntos (después del guión). Mandemos los argumentos `-l` y `-a` a `ls`:

```
eduardo@ubuntu: /tmp
File Edit View Search Terminal Help
eduardo@ubuntu: /tmp$ man 1 ls
eduardo@ubuntu: /tmp$ ls -l -a
total 124
drwxrwxrwt 13 root    root    69632 Sep 30 11:00 .
drwxr-xr-x 28 root    root    4096 Jun 20 11:05 ..
-rw----- 1 eduardo eduardo   0 Sep 30 10:58 config-err-Ikawta
drwx----- 2 eduardo eduardo  4096 Sep 30 10:58 .esd-1000
drwx----- 2 lightdm lightdm  4096 Sep 30 10:58 .esd-125
drwxrwxrwt 2 root    root    4096 Sep 30 10:57 .font-unix
drwxr-xr-x 2 tomcat6 tomcat6  4096 Sep 30 10:57 hspcrfdata_tomcat6
drwxrwxrwt 2 root    root    4096 Sep 30 10:58 .ICE-unix
drwx----- 3 root    root    4096 Sep 30 10:57 systemd-private-9d56ffabacfb492a9b5dafa75cfd0f95-c
olord.service-Foubc5
drwx----- 3 root    root    4096 Sep 30 10:57 systemd-private-9d56ffabacfb492a9b5dafa75cfd0f95-r
tkit-daemon.service-nM99Gf
drwxrwxrwt 2 root    root    4096 Sep 30 10:57 .Test-unix
drwxr-xr-x 2 tomcat6 root    4096 Sep 30 10:57 tomcat6-tomcat6-tmp
-r--r--r-- 1 root    root    11 Sep 30 10:57 .X0-lock
drwxrwxrwt 2 root    root    4096 Sep 30 10:57 .X11-unix
drwxrwxrwt 2 root    root    4096 Sep 30 10:57 .XIM-unix
eduardo@ubuntu: /tmp$
```

Shell Scripts

Podemos combinar los argumentos. Por ejemplo:

```
$ ls -la
```

El resultado es el mismo.

Shell Scripts

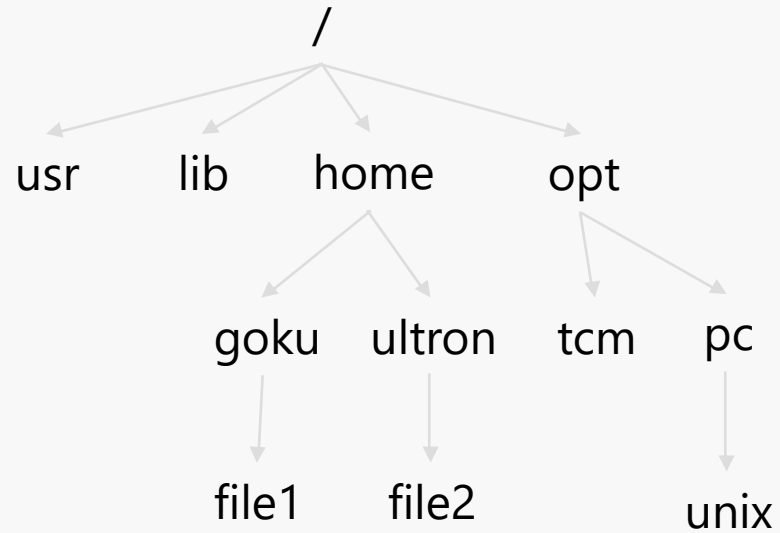
El sistema de archivos

Antes de continuar, debemos primero **ubicarnos**. Todos recordarán en Windows una ruta como esta:

C:\Users\Eduardo

Shell Scripts

- En Linux, el sistema de archivos es jerárquico y unificado:



Shell Scripts

Fíjense que todo empieza en una raíz (/). A esta carpeta la llamamos **root**.

Por ejemplo, la ruta del archivo file2 es:

`/home/ultron/file2`

Todo archivo o carpeta está dentro de esta jerarquía, incluso si es un dispositivo externo (como un pen drive, etc)

Shell Scripts

Comandos para desplazarnos dentro del sistema de archivos:

1. **pwd (print working directory):** para saber la carpeta en la que estamos
2. **ls (list):** muestra el contenido del directorio actual. También podemos especificar un directorio en particular (como argumento).
3. **cd (change directory):** nos permite cambiar de directorio. Por ejemplo:

Shell Scripts

Rutas Absolutas vs Relativas

- Con el comando **cd** (y otros más), podemos usar rutas **absolutas**
 - **relativas:**
 - ***Ruta absoluta:*** empieza con / (desde el directorio raíz).
 - ***Ruta relativa:*** no empieza con /.

Shell Scripts

Rutas relativas

Al usar ruta relativa, el directorio actual de trabajo, es concatenado a la ruta relativa (eduardo), para obtener la **ruta absoluta**. Por ejemplo:

Directorio de trabajo	<code>/home/ed/carpeta</code>
------------------------------	-------------------------------

Ruta relativa:	<code>tarea/archivo1</code>
-----------------------	-----------------------------

Ruta real	<code>/home/ed/carpeta/tarea/archivo1</code>
------------------	--

Shell Scripts

Directorios especiales

- Todo carpeta (**directorio** de aquí en adelante), tiene siempre al menos dos directorios:
 1. *.: este directorio apunta al mismo directorio que lo contiene*
 2. *.. *: este directorio apunta al directorio padre del directorio actual**

Shell Scripts

Comandos básicos

En esta sección aprenderemos comandos básicos. El primero es el comando **man** que nos da la descripción de otros comandos:

man <capítulo> <palabra clave>

Shell Scripts

- Ayuda de Linux este separa en 9 capítulos:

1. *Programas*
2. *Llamadas al sistema*
3. *Rutinas de librerías*
4. *Archivos especiales*
5. *Formato de archivos*
6. *Juegos*
7. *Paquetes Macro*
8. *Comandos de administrador de sistema*
9. *Rutinas del kernel*

Shell Scripts

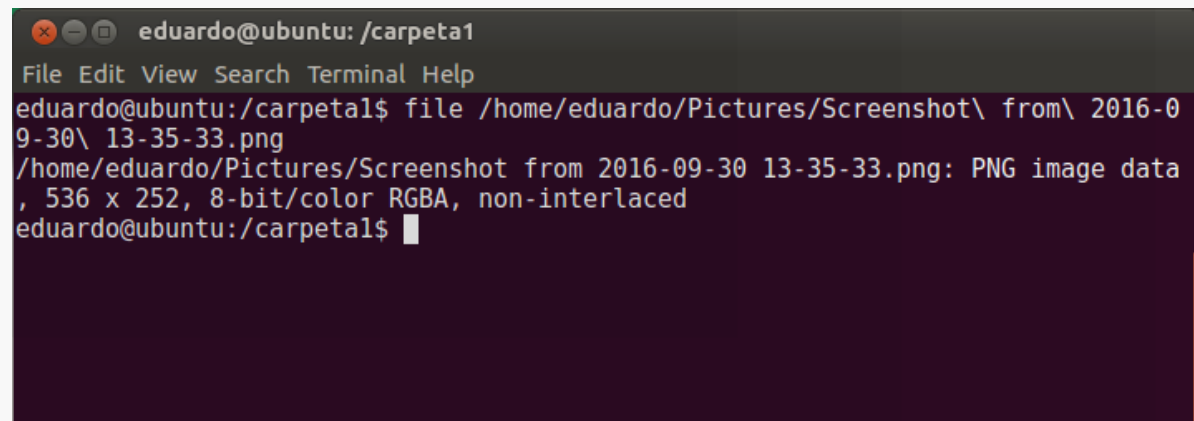
Ver contenido y tipo de archivos

Una vez que ya podemos ubicarnos, vamos a ver dos comandos útiles:

1. **less**: usado para ver el contenido de un archivo
2. **file**: usado para saber el tipo de archivo

Shell Scripts

Para saber el tipo de archivo, usamos **file <nombre archivo>**

A terminal window with a dark background and light text. The title bar shows 'eduardo@ubuntu: /carpetal'. The menu bar includes 'File Edit View Search Terminal Help'. The command 'file /home/eduardo/Pictures/Screenshot\ from\ 2016-09-30\ 13-35-33.png' has been entered and executed. The output shows the file is a PNG image with dimensions 536 x 252, 8-bit color, and non-interlaced data. The prompt 'eduardo@ubuntu: /carpetal\$' is visible at the bottom.

```
eduardo@ubuntu: /carpetal$ file /home/eduardo/Pictures/Screenshot\ from\ 2016-09-30\ 13-35-33.png
/home/eduardo/Pictures/Screenshot from 2016-09-30 13-35-33.png: PNG image data
, 536 x 252, 8-bit/color RGBA, non-interlaced
eduardo@ubuntu: /carpetal$
```

Este comando puede reconocer la mayoría de archivos más utilizados como imágenes, archivos de texto, etc.

Un tour rápido por el sistema de archivos

1. */* : el directorio raíz
2. */etc* : archivos de configuración del sistema (*passwd*, *fstab*, *hosts*, etc).
3. */boot*: archivos para inicializar el sistema operativo
4. */sbin*, */bin*: archivos ejecutables
5. */usr*: contiene variedad de cosas que apoyan
6. */usr/local*: usualmente donde va el software instalado
7. */var*: archivos que cambian a medida que corre el sistema
8. */lib*: librerías compartidas (.a y .so)
9. */home*: directorio personal de los usuarios.
10. */media*, */mnt*: punto de montaje de dispositivos externos.

Shell Scripts

Manipular archivos

Ahora que sabemos desplazarnos, aprenderemos como manipular los archivos y directorios.

Para **copiar archivos**, usamos **cp**:

```
cp <ruta de archivo a copiar> <ruta de destino>
```

Las rutas pueden ser relativas o absolutas, archivos o directorios. Veamos ejemplos.

Shell Scripts

Mover o renombrar

- Para **mover o renombrar** archivos o directorios, usamos el comando **mv** (**move**):

```
mv <ruta a mover> <ruta de destino>
```

- Así mismo, las rutas pueden ser relativas o absolutas, directorios o archivos. Por ejemplo:

Shell Scripts

Removiendo archivos o directorios

- Para remover archivos o directorios llenos, usamos el comando **rm (remove)**:

```
rm <archivo>
```

- Para remover directorios **vacíos**, usamos el comando **rmdir (remove)**:

```
rmdir <directorio>
```

Shell Scripts

Creando directorios

- Para crear directorios, usamos el comando **mkdir** (make directory):

`mkdir <nombre>`

- Nombre puede ser ruta relativa o absoluta.

Shell Scripts

Una pequeña nota sobre los *wildcards*

- Imaginen que tenemos que borrar 10000 imágenes .jpg...¿Lo haríamos así?

```
rm imagen1.jpg imagen2.jpg ... imagen10000.jpg
```

- Podríamos... Pero escribir todo eso se vuelve tedioso. Sin embargo, hay una alternativa:

```
rm *.jpg
```

Shell Scripts

- El símbolo * es uno de los llamados wildcards, y significa "cualquier texto". Entonces nuestro comando dice:

Remover todos los archivos (de cualquier nombre), que terminen en .jpg

- Existen otros wildcards son mostrados a continuación.

¡PODEMOS USAR WILDCARDS CON LOS COMANDOS!

Shell Scripts

Sección	Contenidos
*	Todo el texto
?	Hace match con cualquier caracter.
[caracteres]	<p>Hace match con cualquiera de los caracteres listados. Podemos especificar clases de caracteres:</p> <ul style="list-style-type: none">[alnum:] → caracteres alfanumericos[alpha:] → caracteres alfabeticos[digit:] → digitos[upper:] → mayusculas[lower:] → minusculas
[!caracteres]	Hace match con cualquiera de los caracteres no listados

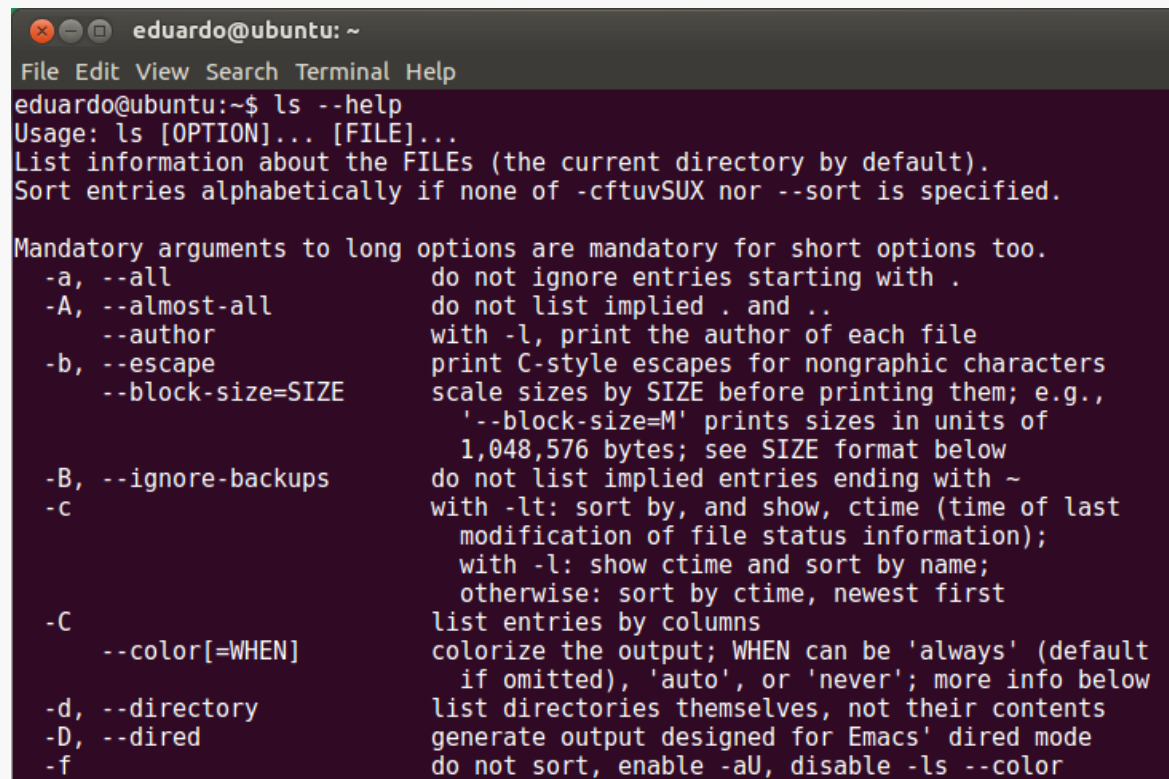
Ejemplos

*	Todos los archivos
ed*	Todos los archivos que empiezan con ed
img*.jpg	Todos los archivos que empiecen con img y terminen en .jpg
Libro???	Palabra Libro seguido de exactamente 3 caracteres
[abc]*	Todos los que comienzan con a, b o c seguido de cualquier texto
[:upper:]*	Todos los que comienzan con mayúsculas seguidos de cualquier texto
back[[:digit:]].*	Todos los que comiencen con "back", seguido de un dígito, punto y cualquier extensión
file[!a]	Todos los que empiecen con file, pero que NO tengan a .

Ciertos comandos útiles para cuando nos perdamos

La mayoría de comandos soportan el argumento `-h` o `--help`:

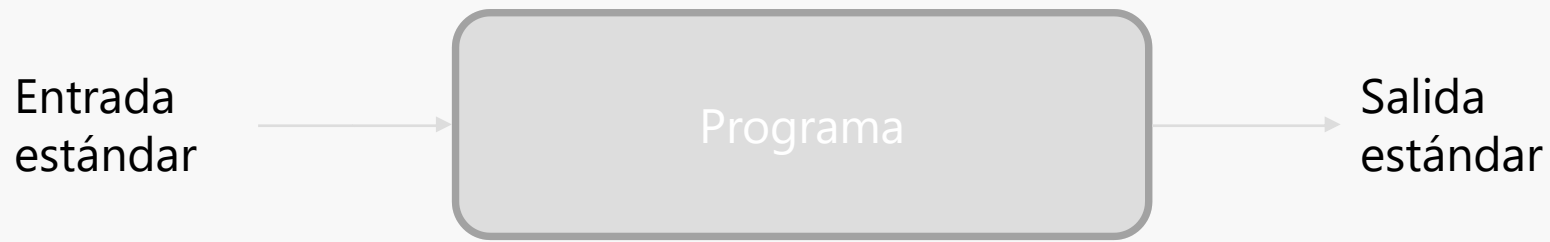
`ls --help`

A terminal window titled 'eduardo@ubuntu: ~' showing the output of the 'ls --help' command. The output includes usage information, a list of mandatory arguments to long options, and a list of short options with their descriptions. The terminal has a dark background with light-colored text.

```
eduardo@ubuntu: ~  
File Edit View Search Terminal Help  
eduardo@ubuntu:~$ ls --help  
Usage: ls [OPTION]... [FILE]...  
List information about the FILEs (the current directory by default).  
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.  
  
Mandatory arguments to long options are mandatory for short options too.  
-a, --all                do not ignore entries starting with .  
-A, --almost-all        do not list implied . and ..  
    --author              with -l, print the author of each file  
-b, --escape             print C-style escapes for nongraphic characters  
    --block-size=SIZE     scale sizes by SIZE before printing them; e.g.,  
                          '--block-size=M' prints sizes in units of  
                          1,048,576 bytes; see SIZE format below  
-B, --ignore-backups     do not list implied entries ending with ~  
-c                       with -lt: sort by, and show, ctime (time of last  
                          modification of file status information);  
                          with -l: show ctime and sort by name;  
                          otherwise: sort by ctime, newest first  
-C                       list entries by columns  
    --color[=WHEN]       colorize the output; WHEN can be 'always' (default  
                          if omitted), 'auto', or 'never'; more info below  
-d, --directory          list directories themselves, not their contents  
-D, --dired              generate output designed for Emacs' dired mode  
-f                       do not sort, enable -aU, disable -ls --color
```

Shell Scripts

Entrada y Salida



Todo programa recibe información por la **ENTRADA ESTÁNDAR** (usualmente el teclado), y envía información por la **SALIDA ESTÁNDAR** (usualmente la pantalla del shell).

Shell Scripts

Redirección de entrada y Salida

Nosotros podemos hacer que la salida de un programa, en vez de mostrarse en pantalla, se escriba a un archivo:

- `>` : *redirección de salida con truncamiento del archivo*
- `>>` : *redirección con anexación al archivo*

*Por ejemplo, para escribir la salida de **ls** al archivo **mi_archivo**:*

```
ls > mi_archivo
```

Si **mi_archivo** no existe, es creado. Si volvemos a ejecutar la misma línea, cualquier contenido que tenía será borrado, y se escribirá el nuevo contenido.

Si queremos evitar que se borre lo que ya existe en el archivo, hacemos redirección con **anexación**:

```
ls >> mi_archivo
```


También podemos redireccionar la entrada con el símbolo <:

```
sort < mi_archivo
```

El programa sort (para ordenar el texto), va a leer del archivo mi_archivo. Finalmente, podemos combinar las redirecciones:

```
sort < mi_archivo > archivo_ordenado
```

¿Qué estamos haciendo aquí?

Shell Scripts

Pipes

A veces, para resolver una tarea, necesitamos hacer uso de varios programas. El resultado de uno lo procesará otro. Para esto usamos los pipes.



Los pipes nos permiten **conectar** la salida estándar de un programa con la entrada estándar de otro.

Por ejemplo:

```
ls -l | grep carpeta
```

En este ejemplo, la salida de **ls** se la pasamos a la entreda de **grep** (que lo usamos para buscar texto), para que busque la palabra carpeta en lo que arrojó **ls**.

Shell Scripts

Filtros

Son programas que reciben algo desde la entrada estándar, realizan una operación, y lo devuelven a la salida estándar. Algunos son:

1. **sort:** ordena las líneas de un texto
2. **uniq:** remueve las líneas duplicadas
3. **grep:** busca el patrón de texto requerido
4. **head:** muestra las primeras líneas del texto
5. **tail:** muestra las últimas líneas del texto
6. **fmt:** muestra el texto con formato
7. **pr:** divide el texto en paginas, con cabeceras y pies de página

Shell Scripts

Otros comandos útiles

ln -> crea un link a un archivo o carpeta

df -> muestra el espacio libre en disco

touch -> cambia la fecha de modificación de un archivo (lo crea si no existe)

time -> nos da la hora

date -> nos da la fecha

Shell Scripts

Cambiando permisos

Para esto usamos la herramienta chmod:

```
chmod <permisos> <archivo>
```

Shell Scripts

Variables

En el shell, nosotros podemos almacenar información en variables.

Para **crear** una variables, hacemos los siguiente en el shell

```
$ mivar='/etc'
```

Esto creará la variable **mivar** con el contenido `"/etc"`. Para hacer **referencia** a una variables, usamos el signo de dólar:

```
$ echo $mivar  
/etc
```

Podemos usar estas variables dentro de string:

```
eduardo@ubuntu:~$ mi_text='mundo'
```

```
eduardo@ubuntu:~$ echo "hola $mi_text"
```

```
hola mundo
```

También podemos poner la variable dentro de llaves, para saber que hacemos referencia a una variable:

```
eduardo@ubuntu:~$ mi_text='mundo'
```

```
eduardo@ubuntu:~$ echo "hola ${mi_text}"
```

```
hola mundo
```


Shell Scripts

Bash Scripting

Ahora que ya conocemos nuestra herramienta (el shell), nos enfocaremos en los *bash scripts*.

¿Qué es un bash script?

Cuando necesitamos realizar una tarea que requiere muchos comandos, los podemos agrupar en un archivo de texto, y luego ejecutarlo en el shell. Todo script empieza con la siguiente línea:

```
#!/bin/sh
```

Por ejemplo, creemos el siguiente script, en el archivo **mi_script.sh**:

```
#!/bin/bash
```

```
echo "hola bash script"
```

Necesitamos hacerlo ejecutable, con el siguiente comando:

```
$ chmod +x mi_script.sh
```

Para ejecutarlo:

```
$ ./mi_script.sh
```

Asi como en el shell, las variables se definen así:

NOMBRE=valor

Para usar el valor de la variable, usamos el signo de \$:

#!/bin/bash

QUIEN="bash script"

echo "hola \$QUIEN"

Cuando ponemos entre comillas, se toma como un string literal (se respetan saltos de línea, etc)

Podemos también asignar el resultado de comandos:

NOMBRE=\$(comando arg1 arg2 ...)

Esto ejecuta el comando en un shell distinto, y lo asigna a la variable

Shell Scripts

Argumentos

Podemos mandar argumentos al script. Estos se almacenan en las variables \$1, \$2, etc. Podemos modificar nuestro script:

```
#!/bin/bash
```

```
echo "hola $1"
```

Y los ejecutamos así:

```
$ ./mi_script "bash script"
```

Shell Scripts

Hay ciertas variables especiales para manejar los argumentos:

- `$0`: contiene el nombre del script, en nuestro caso **`./mi_script`**
- `$1, $2, ...` : los argumentos enviados al script
- `$#`: contiene el número de argumentos
- `$*`: contiene todos los argumentos
- `$$`: ID proceso script actual

Shell Scripts

Funciones

Para scripts más largos, podemos crear funciones, de esta forma:

```
function <nombre función> {  
    comando1  
    comando2  
    ...  
}
```

Cambiemos nuestro script para que usen funciones...

```
#!/bin/bash

#función...

function mostrar_mensaje{

    echo "hola $1"

}

#script principal


mostrar_mensaje
```

Noten que los comentarios empiezan con #.

Así como en otros lenguajes, las variables declaradas dentro de una función son **locales** (usando la palabra **local**), y fuera de ella, **globales**.

```
#!/bin/bash

QUIEN="bash"

function mensaje {
    local QUIEN_MAS
    QUIEN_MAS = "world"
    echo "Hello $1"
    echo "Hello $QUIEN"
    echo "Hello $QUIEN_MAS"
}

mensaje

echo "bye $QUIEN"
echo "bye $1"
echo "bye $QUIEN_MAS"
```

- Si ejecutamos...

```
eduardo@ubuntu:~$ ./mi_script "bash script"
Hello
Hello bash
Hello world
bye bash
bye bash script
bye
```

En la última línea, no sale el nombre, por que tratamos de usar una variable local **fuera de la función**.

Shell Scripts

Condicionales

Nos permiten evaluar condiciones. La sintaxis es

```
if [ condicion] ; then
```

```
    comandos
```

```
elif [condicion2] ; then
```

```
    comandos
```

```
else
```

```
    comandos
```

```
fi
```

← indica fin de bloque if

Las condiciones van seguidas de punto y coma

¿Como hacemos las comparaciones?

Para comparaciones numéricas:

String	Numeric	True if
$x = y$	$x \text{ -eq } y$	x is equal to y
$x \neq y$	$x \text{ -ne } y$	x is not equal to y
$x < y$	$x \text{ -lt } y$	x is less than y
$x \leq y$	$x \text{ -le } y$	x is less than or equal to y
$x > y$	$x \text{ -gt } y$	x is greater than y
$x \geq y$	$x \text{ -ge } y$	x is greater than or equal to y
$\text{-n } x$	–	x is not null
$\text{-z } x$	–	x is null

Para comparaciones de archivos (!)

Operator	True if
<code>-d file</code>	<i>file</i> exists and is a directory
<code>-e file</code>	<i>file</i> exists
<code>-f file</code>	<i>file</i> exists and is a regular file
<code>-r file</code>	You have read permission on <i>file</i>
<code>-s file</code>	<i>file</i> exists and is not empty
<code>-w file</code>	You have write permission on <i>file</i>
<code>file1 -nt file2</code>	<i>file1</i> is newer than <i>file2</i>
<code>file1 -ot file2</code>	<i>file1</i> is older than <i>file2</i>

Por ejemplo:

```
#!/bin/bash

if [ $1 -eq 0 ]; then
    echo "hola"
else
    echo "chao"
fi
```

Imprimirá hola si el argumento es 0, y chao si es 1.

Para comparar strings, hay que tener cuidado:

```
#!/bin/bash

if [ $1 == "bash" ]; then
    echo "hola $1"
else
    echo "chao $1"
fi
```

¡Usamos el operador ==!

Shell Scripts

Lazos

Podemos tener lazos en los scripts, para realizar tareas repetitivas. Para el lazo **for**:

```
for mi_var in {inicio..fin}; do
    comandos
comandos $mi_var
done
```

Veamos un ejemplo.

Shell Scripts

```
for mi_var in archivo1 archivo2; do  
    comandos  
    comandos $mi_var  
done
```

```
for <variable> in ${comando_shell} do  
    comandos  
done
```

Asumamos que tenemos una carpeta con cuatro archivo llamados file1, file2, file3 y file4, además de otros archivos y queremos borrar **solo** los archivos que empiezan con file seguidos de un dígito. Podemos entonces:

```
#!/bin/bash

for archivo in file[[:digit:]] ; do
    rm $archivo
done
```

Shell Scripts

Podemos tener lazos en los scripts, para realizar tareas repetitivas.

Para el lazo **while**:

```
While <condicion> ; do  
    comandos  
done
```

Veamos un ejemplo.

Asumamos que tenemos una carpeta con cuatro archivo llamados file1, file2, file3 y file4, además de otros archivos y queremos borrar **solo** los archivos que empiezan con file seguidos de un dígito. Podemos entonces:

```
#!/bin/bash

for archivo in file[[:digit:]] ; do
    rm $archivo
done
```

Ejemplo, crear cuatro directorios llamados dir0, dir1, dir2, dir3:

```
#!/bin/bash

i=0

while [ $i -lt 4 ];
do
    mkdir "dir$i"
    ((i++))
done

ls -l
```

Shell Scripts

Entrada y salida

Para leer texto desde el teclado, usamos **read**, y para imprimir algo en pantalla, **echo**. Por ejemplo, el siguiente script lee el teclado e imprime lo leído en pantalla.

```
#!/bin/bash

echo -n "Ingrese su nombre: "

read user_name

echo $user_name
```

Shell Scripts

- Podemos usar redirección también. Por ejemplo, leer un archivo línea por línea:

```
for read linea;  
do  
    echo $linea  
done < archivo
```