

# Taller Ext2

Departamento de Computación, FCEyN,  
Universidad de Buenos Aires, Buenos Aires, Argentina

Sistemas Operativos, Segundo Cuatrimestre 2024

## (2) Presentación Taller

- Vamos a programar un Sistema de Archivos ext2.

## (2) Presentación Taller

- Vamos a programar un Sistema de Archivos ext2.
- ¿Qué debemos conocer y tener para lograrlo?

## (2) Presentación Taller

- Vamos a programar un Sistema de Archivos ext2.
- ¿Qué debemos conocer y tener para lograrlo?
  - Lo que aprendimos en las clases teórica y práctica sobre ext2.

## (2) Presentación Taller

- Vamos a programar un Sistema de Archivos ext2.
- ¿Qué debemos conocer y tener para lograrlo?
  - Lo que aprendimos en las clases teórica y práctica sobre ext2.
  - Un disco al que podemos acceder a cualquiera de sus bloques.

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:

```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```



### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:

```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco?

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:

```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco?

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:

```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque?

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:


```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque?

### (3) El disco

- ¿Qué es? Un montón de bytes agrupados en bloques.
- A cada bloque se accede con su LBA (Logical Block Addressing).
- API de HDD:

```
int read(unsigned int lba, unsigned char * buffer);  
int write(unsigned int lba, unsigned char * buffer);
```

- ¿Qué tamaño tiene el disco? A priori no se conoce.
- ¿Qué tamaño tiene cada bloque? 1024 bytes.
- ¿Por dónde se empieza? 

## (4) Boot block

- *Bloque de Booteo o Master Boot Record*

## (4) Boot block

- *Bloque de Booteo o Master Boot Record*
- Está en la primera parte del disco. Es un espacio reservado de 1024 bytes.

Structure of a classical generic MBR

Address		Description		Size (bytes)
Hex	Dec			
+000h	+0	Bootstrap code area		446
+1BEh	+446	Partition entry #1	Partition table (for primary partitions)	16
+1CEh	+462	Partition entry #2		16
+1DEh	+478	Partition entry #3		16
+1EEh	+494	Partition entry #4		16
+1FEh	+510	55h	Boot signature <sup>[a]</sup>	2
+1FFh	+511	AAh		
Total size: 446 + 4×16 + 2				512

## (4) Boot block

- *Bloque de Booteo o Master Boot Record*
- Está en la primera parte del disco. Es un espacio reservado de 1024 bytes.

Structure of a classical generic MBR

Address		Description	Size (bytes)
Hex	Dec		
+000h	+0	Bootstrap code area	446
+1BEh	+446	Partition entry #1	16
+1CEh	+462	Partition entry #2	16
+1DEh	+478	Partition entry #3	16
+1EEh	+494	Partition entry #4	16
+1FEh	+510	55h	2
+1FFh	+511	AAh	
Total size: 446 + 4×16 + 2			512

- Sólo contiene los datos necesarios para iniciar la máquina y nada más (esto es así en TODOS los sistemas de archivos).



## (5) Partición de ext2

- Llegamos hasta donde empieza el primer grupo de bloques.


## (5) Partición de ext2

- Llegamos hasta donde empieza el primer grupo de bloques.
- El superblock: tiene información de TODO el sistema de archivos.

## (5) Partición de ext2

- Llegamos hasta donde empieza el primer grupo de bloques.
- El superblock: tiene información de TODO el sistema de archivos.
- ¿En qué parte de la partición está?

## (5) Partición de ext2

- Llegamos hasta donde empieza el primer grupo de bloques.
- El superblock: tiene información de TODO el sistema de archivos.
- ¿En qué parte de la partición está?
- A partir del byte 1024. Independientemente del tamaño del bloque. 

## (6) Superblock

```
struct Ext2FSSuperblock {
__le32 s_inodes_count;      /* Contador de inodos */
__le32 s_blocks_count;     /* Contador de bloques */
__le32 s_r_blocks_count;   /* Contador de bloques reservados */
__le32 s_free_blocks_count; /* Contador de bloques libres */
__le32 s_free_inodes_count; /* Contador de inodos libres */
__le32 s_first_data_block; /* Primer bloque de Datos */
__le32 s_log_block_size;   /* Tamano del bloque */
...
__le32 s_blocks_per_group; /* Cantidad de bloques por grupo */
...
__le32 s_inodes_per_group; /* Cantidad de inodos por grupos */
...
__le16 s_magic;            /* Firma magica — identifica el S.A. */
__le32 s_first_ino;        /* Primer inodo no reservado */
__le16 s_inode_size;       /* Tamano de la estructura del Inodo */
```

## (7) Estructura de Ext2

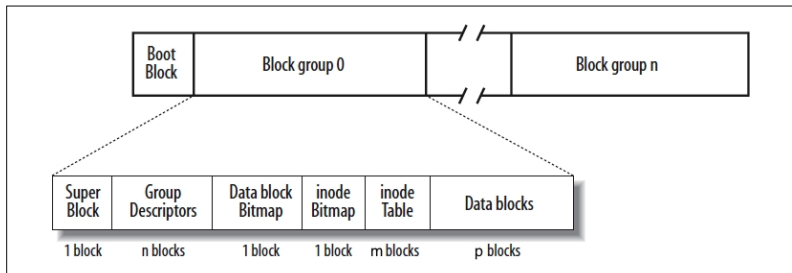
- ¿Dónde está mi archivo `/home/krypton.gis`?

## (7) Estructura de Ext2

- ¿Dónde está mi archivo `/home/krypton.gis`?
- Recordemos que en ext2 todo está representado por Inodos.  
¿Cuál es el inodo de mi archivo?

## (7) Estructura de Ext2

- ¿Dónde está mi archivo `/home/krypton.gis`?
- Recordemos que en ext2 todo está representado por Inodos.  
¿Cuál es el inodo de mi archivo?
- Supongamos que está en el Inodo 4483.





## (8) Estructura de Ext2

- Tenemos que calcular en qué Block Group se encuentra.

## (8) Estructura de Ext2

- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.

## (8) Estructura de Ext2

- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.
- Esa información está en el superbloque.

## (8) Estructura de Ext2

- Tenemos que calcular en qué Block Group se encuentra.
- Para eso necesitamos averiguar cuántos inodos hay por Block Group.
- Esa información está en el superbloque.
- Tenemos que hacer la división entre nuestro número de inodo y la cantidad de inodos. Eso nos va a determinar el Block Group.

## (9) Estructura de Ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por  $m$  bloques de tamaño block size. En cada bloque habrá un conjunto de inodos.

## (9) Estructura de Ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por  $m$  bloques de tamaño block size. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.

## (9) Estructura de Ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por  $m$  bloques de tamaño block size. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.

## (9) Estructura de Ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por  $m$  bloques de tamaño block size. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.
- Una vez conseguido, tenemos que leer ese bloque de disco y luego del conjunto de inodos, conseguir el que nos interesa.



## (9) Estructura de Ext2

- Recordemos que cada Block Group tiene una tabla de inodos. Esta tabla está constituida por  $m$  bloques de tamaño block size. En cada bloque habrá un conjunto de inodos.
- Tenemos que averiguar que número de inodo corresponde de nuestra tabla de inodo. Para eso usamos el módulo con la cantidad de inodos.
- Luego tenemos que localizar en qué bloque está nuestro inodo a buscar.
- Una vez conseguido, tenemos que leer ese bloque de disco y luego del conjunto de inodos, conseguir el que nos interesa.
- Funciones utiles: `blockgroup_for_inode`, `blockgroup_inode_index`.

## (10) Inodo

- La representación de un archivo.

## (10) Inodo

- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.

## (10) Inodo


- La representación de un archivo.
- Un archivo puede ser un archivo regular, un directorio, un pipe, un socket, un device, etc.
- A bajo nivel, en este taller, es un struct de FSInode.

## (11) FSInode

```
struct Ext2FSInode {  
    unsigned short mode;  
    unsigned short uid;  
    unsigned int size;  
    unsigned int atime;  
    unsigned int ctime;  
    unsigned int mtime;  
    unsigned int dtime;  
    unsigned short gid;  
    unsigned short links_count;  
    unsigned int blocks;  
    unsigned int flags;  
    unsigned int os_dependant_1;  
    unsigned int block[15];  
    unsigned int generation;  
    unsigned int file_acl;  
    unsigned int directory_acl;  
    unsigned int faddr;  
    unsigned int os_dependant_2[3];  
}
```



## (11) FSInode

```
struct Ext2FSInode {  
    unsigned short mode;  
    unsigned short uid;  
    unsigned int size;  
    unsigned int atime;  
    unsigned int ctime;  
    unsigned int mtime;  
    unsigned int dtime;  
    unsigned short gid;  
    unsigned short links_count;  
    unsigned int blocks;  
    unsigned int flags;  
    unsigned int os_dependant_1;  
    unsigned int block[15];  
    unsigned int generation;  
    unsigned int file_acl;  
    unsigned int directory_acl;  
    unsigned int faddr;  
    unsigned int os_dependant_2[3];  
}
```

- ¿Dónde están los datos? 



## (11) FSInode

```
struct Ext2FSInode {  
    unsigned short mode;  
    unsigned short uid;  
    unsigned int size;  
    unsigned int atime;  
    unsigned int ctime;  
    unsigned int mtime;  
    unsigned int dtime;  
    unsigned short gid;  
    unsigned short links_count;  
    unsigned int blocks;  
    unsigned int flags;  
    unsigned int os_dependant_1;  
    unsigned int block[15];  
    unsigned int generation;  
    unsigned int file_acl;  
    unsigned int directory_acl;  
    unsigned int faddr;  
    unsigned int os_dependant_2[3];  
}
```

- ¿Dónde están los datos? 
- ¿Dónde está el nombre del archivo? 

## (11) FSInode

```
struct Ext2FSInode {  
    unsigned short mode;  
    unsigned short uid;  
    unsigned int size;  
    unsigned int atime;  
    unsigned int ctime;  
    unsigned int mtime;  
    unsigned int dtime;  
    unsigned short gid;  
    unsigned short links_count;  
    unsigned int blocks;  
    unsigned int flags;  
    unsigned int os_dependant_1;  
    unsigned int block[15];  
    unsigned int generation;  
    unsigned int file_acl;  
    unsigned int directory_acl;  
    unsigned int faddr;  
    unsigned int os_dependant_2[3];  
}
```

- ¿Dónde están los datos? 
- ¿Dónde está el nombre del archivo?  A nivel de usuario no se hace referencia a números de inodos.



# (11) FSInode

```
struct Ext2FSInode {
    unsigned short mode;
    unsigned short uid;
    unsigned int size;
    unsigned int atime;
    unsigned int ctime;
    unsigned int mtime;
    unsigned int dtime;
    unsigned short gid;
    unsigned short links_count;
    unsigned int blocks;
    unsigned int flags;
    unsigned int os_dependant_1;
    unsigned int block[15];
    unsigned int generation;
    unsigned int file_acl;
    unsigned int directory_acl;
    unsigned int faddr;
    unsigned int os_dependant_2[3];
}
```

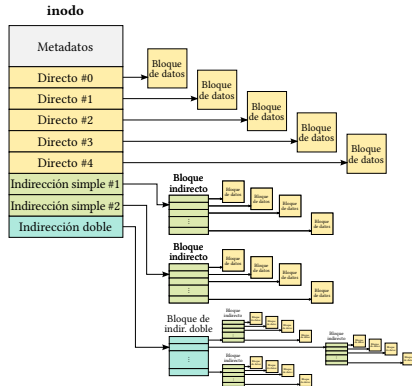
- ¿Dónde están los datos? ⚠
- ¿Dónde está el nombre del archivo? ⚠ A nivel de usuario no se hace referencia a números de inodos.
- ¿El inodo directorio qué struct usa? ⚠

## (12) Inodo - Datos

- 15 punteros a bloques con distintos sabores:

## (12) Inodo - Datos

- 15 punteros a bloques con distintos sabores:
  - 12 punteros a bloques de datos directos
  - 1 puntero indirecto a bloque de datos
  - 1 puntero con una doble indirección a bloque de datos
  - 1 puntero con una triple indirección a bloque de datos
- Solo vamos a implementar hasta la doble indirección ⚠



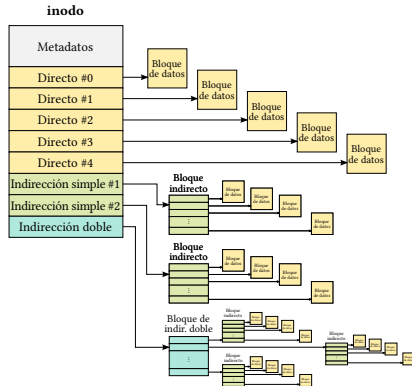
# (12) Inodo - Datos

- 15 punteros a bloques con distintos sabores:

- 12 punteros a bloques de datos directos
- 1 puntero indirecto a bloque de datos
- 1 puntero con una doble indirección a bloque de datos
- 1 puntero con una triple indirección a bloque de datos

- Solo vamos a implementar hasta la doble indireccion ⚠

- ¿Son punteros a direcciones de memoria? ⚠



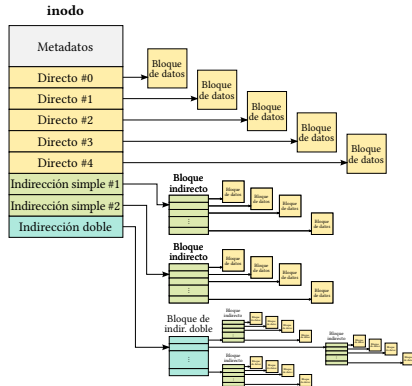
# (12) Inodo - Datos

- 15 punteros a bloques con distintos sabores:

- 12 punteros a bloques de datos directos
- 1 puntero indirecto a bloque de datos
- 1 puntero con una doble indirección a bloque de datos
- 1 puntero con una triple indirección a bloque de datos

- Solo vamos a implementar hasta la doble indireccion ⚠

- ¿Son punteros a direcciones de memoria? ⚠
- ¿Los bloques a los que apuntan, están en memoria o en disco? ⚠



## (13) Inodo - Directorio

- Es un Inodo IGUAL que cualquier otro.

## (13) Inodo - Directorio


- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.

## (13) Inodo - Directorio

- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.
- Entonces, ¿dónde está la lista de archivos de mi directorio?



## (13) Inodo - Directorio

- Es un Inodo IGUAL que cualquier otro.
- Es decir, tiene la misma estructura `Ext2FSInode`.
- Entonces, ¿dónde está la lista de archivos de mi directorio?
- En los bloques de datos. 


## (14) DirEntry

```
struct Ext2FSDirEntry {  
    unsigned int inode;  
    unsigned short record_length;  
    unsigned char name_length;  
    unsigned char file_type;  
    char name[];  
};
```

- **Los datos** del Inodo son una lista de structs Ext2FSDirEntry.


## (14) DirEntry

```
struct Ext2FSDirEntry {  
    unsigned int inode;  
    unsigned short record_length;  
    unsigned char name_length;  
    unsigned char file_type;  
    char name[];  
};
```

- **Los datos** del Inodo son una lista de structs Ext2FSDirEntry.
- Cada struct tiene tamaño variable. 

- ¡Puede haber un caso borde! 


## (15) Inodo - Directorio

- ¡Puede haber un caso borde! 
- Puede pasar que el struct de DirEntry quede dividido en dos bloques.


## (16) Inodo - Directorio

- ¿Qué pasa si busco un archivo que no existe en este directorio? 

## (16) Inodo - Directorio


- ¿Qué pasa si busco un archivo que no existe en este directorio? 

## (16) Inodo - Directorio


- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.




## (16) Inodo - Directorio

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.


## (16) Inodo - Directorio

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.
- ¿Qué condicion usamos de corte?.

## (16) Inodo - Directorio

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.
- ¿Qué condicion usamos de corte?.

## (16) Inodo - Directorio

- ¿Qué pasa si busco un archivo que no existe en este directorio? 
- Debemos buscar en cada direntry que se encuentre en el directorio hasta llegar al final.
- ¿Qué condicion usamos de corte?.
- El campo size del inodo nos dice la cantidad de bytes que usa ese inodo.

## (17) Enunciado

Completar la implementación de los siguientes métodos:

- 1 `Ext2FSInode* load_inode(inode_number)`
- 2 `unsigned int get_block_address(inode, block_number)`
- 3 `Ext2FSInode* get_file_inode_from_dir_inode(from, filename)`

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.
- Las estructuras de Ext2FS:



## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.
- Las estructuras de Ext2FS:
  - Ext2FSSuperblock (Superblock)

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.
- Las estructuras de Ext2FS:
  - Ext2FSSuperblock (Superblock)
  - Ext2FSBlockGroupDescriptor (Block Group Descriptor)

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.
- Las estructuras de Ext2FS:
  - Ext2FSSuperblock (Superblock)
  - Ext2FSBlockGroupDescriptor (Block Group Descriptor)
  - Ext2FSInode (Inode)

## (18) ¿Qué parte del código ya está preparada?

- Las clases HDD, MBR y PartitionEntry.
- Parcialmente la clase Ext2FS.
- Las estructuras de Ext2FS:
  - Ext2FSSuperblock (Superblock)
  - Ext2FSBlockGroupDescriptor (Block Group Descriptor)
  - Ext2FSInode (Inode)
  - Ext2FSDirEntry (Directory Entry)

## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:

## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:
  - `read_block(block_address, buffer)`: Lee de disco el bloque de dirección `block_address` y lo coloca en `buffer`. `buffer` es un puntero a char, recordar castear si queremos indexar por algo de mayor tamaño cuando lo vayamos a utilizar, como `int`.

## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:
  - `read_block(block_address, buffer)`: Lee de disco el bloque de dirección `block_address` y lo coloca en `buffer`. `buffer` es un puntero a `char`, recordar castear si queremos indexar por algo de mayor tamaño cuando lo vayamos a utilizar, como `int`.
  - `superblock()`: Devuelve el superbloque.

## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:
  - `read_block(block_address, buffer)`: Lee de disco el bloque de dirección `block_address` y lo coloca en `buffer`. `buffer` es un puntero a char, recordar castear si queremos indexar por algo de mayor tamaño cuando lo vayamos a utilizar, como `int`.
  - `superblock()`: Devuelve el superbloque.
  - `block_group(index)`: Devuelve el descriptor del grupo de bloques del bloque `index`.



## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:
  - `read_block(block_address, buffer)`: Lee de disco el bloque de dirección `block_address` y lo coloca en `buffer`. `buffer` es un puntero a `char`, recordar castear si queremos indexar por algo de mayor tamaño cuando lo vayamos a utilizar, como `int`.
  - `superblock()`: Devuelve el superbloque.
  - `block_group(index)`: Devuelve el descriptor del grupo de bloques del bloque `index`.
  - `blockgroup_for_inode(inode)`: Número de grupo de bloques del inodo.

## (19) ¿Qué parte del código ya está preparada?

- Las funciones auxiliares de Ext2FS:
  - `read_block(block_address, buffer)`: Lee de disco el bloque de dirección `block_address` y lo coloca en `buffer`. `buffer` es un puntero a `char`, recordar castear si queremos indexar por algo de mayor tamaño cuando lo vayamos a utilizar, como `int`.
  - `superblock()`: Devuelve el superbloque.
  - `block_group(index)`: Devuelve el descriptor del grupo de bloques del bloque `index`.
  - `blockgroup_for_inode(inode)`: Número de grupo de bloques del inodo.
  - `blockgroup_inode_index(inode)`: Offset dentro de la tabla de inodos, para el inodo.

## (20) Últimos tips

- Usen la VM de la materia

## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.

## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.


## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.

## (20) Últimos tips


- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.

## (20) Últimos tips


- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.
- Recuerden, los directorios son archivos. 




## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.
- Recuerden, los directorios son archivos. 
- Documentación:


## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.
- Recuerden, los directorios son archivos. 
- Documentación:
  - <http://www.nongnu.org/ext2-doc/ext2.html>

## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.
- Recuerden, los directorios son archivos. 
- Documentación:
  - <http://www.nongnu.org/ext2-doc/ext2.html>
  - <http://e2fsprogs.sourceforge.net/ext2intro.html>

## (20) Últimos tips

- Usen la VM de la materia
- Hagan los ejercicios en el orden dado.
- Descompriman la imagen `hdd.raw.xz` para usarla.
- Hay estructuras para cada tipo.
- Utilicen las funciones auxiliares.
- Recuerden, los directorios son archivos. 
- Documentación:
  - <http://www.nongnu.org/ext2-doc/ext2.html>
  - <http://e2fsprogs.sourceforge.net/ext2intro.html>
  - <http://wiki.osdev.org/Ext2>