



Proyecto 2

26 de octubre del 2023

Principios de Sistemas operativos

GR 01

Mariángeles Carranza Varela – 2021135530

Anthony Jiménez Barrantes – 2021022457

Contenidos

| | |
|--|----|
| Introducción | 3 |
| Descripción del problema (este enunciado) | 3 |
| Definición de estructuras de datos | 6 |
| Descripción detallada y explicación de los componentes principales del programa: | 7 |
| Mecanismo de acceso a archivos | 7 |
| Estructura de directorios internos | 8 |
| Estrategia de administración de espacios libres | 9 |
| Procedimiento de desfragmentación del archivo | 10 |
| Análisis de resultados de pruebas (funcionamiento) | 10 |
| Conclusiones | 21 |

Introducción

El proyecto "star" (Simple Tar) es una implementación de una herramienta de empaquetamiento de archivos que replica las funcionalidades esenciales del comando "tar" en sistemas UNIX. Este programa ofrece la capacidad de crear, listar, extraer, eliminar, actualizar, agregar contenido y desfragmentar archivos en un archivo de formato tar. El proyecto se ha desarrollado cuidadosamente, teniendo en cuenta diversos aspectos importantes.

Para abordar el problema, se han creado estructuras de datos específicas, como "file" para representar archivos individuales, "directory" para gestionar el archivo tar y "LogEntry" para registrar mensajes detallados. El programa utiliza operaciones estándar de acceso a archivos en C y se basa en una estructura de directorios internos eficiente para organizar los archivos en el archivo tar.

La estrategia de administración de espacios libres se basa en la estrategia de primer ajuste, y la desfragmentación se realiza mediante la compactación del archivo tar. Se han realizado pruebas exhaustivas para verificar el funcionamiento de cada función individual y en combinación, con registros detallados proporcionados a través de la opción "--verbose".

Descripción del problema (este enunciado)

El objetivo de este proyecto consiste en programar un emparador de archivos. Este es el tipo de funcionalidad que provee el comando *tar* en ambientes UNIX.

El programa *tar*, es usado para almacenar múltiples archivos en un solo archivo. Dentro de los entornos Unix *tar* aparece como un comando que puede ser ejecutada desde la línea de comandos de una consola de texto o desde un simple terminal. El formato del comando *tar* es, comúnmente

`tar <opciones> <archivoSalida> <archivo1> <archivo2> ... <archivoN>`

donde <archivoSalida> es el archivo resultado y <archivo1>, <archivo2>, etc; son los diferentes archivos que serán "empaquetados" en <archivoSalida>.

Las opciones más comunes son las siguientes:

- -c, --create : crea un nuevo archivo
- -x, --extract : extraer de un archivo
- -t, --list: listar los contenidos de un archivo
- --delete: borrar desde un archivo

- -u, --update: actualiza el contenido del archivo
- -v, --verbose: ver un reporte de las acciones a medida que se van realizando
- -f, --file: empaclar contenidos de archivo, si no está presente asume la entrada estándar.
- -r, --append: agrega contenido a un archivo
- -p, --pack: desfragmenta el contenido del archivo (no presente en tar)

Ejemplos

1. Si queremos empaclar un archivo llamado "index.html" y guardar los datos en "html-paq.tar", lo haríamos con la instrucción

```
tar -cvf html-paq.tar index.html
```

2. Si queremos desempaquetar todo el contenido de un archivo llamado xxx.tar podemos utilizar un comando como este

```
tar -xvf xxx.tar
```

3. Para archivar el contenido de tres archivos doc1.txt, doc2.txt y data.dat

```
tar -cvf foo.tar doc1.txt doc2.txt data.dat
```

4. Si ahora se desea eliminar el contenido del archivo data.dat se ejecutaría

```
tar --delete -vf foo.tar data.dat
```

5. Para agregar ahora un nuevo archivo test.doc a foo.tar se ejecutaría

```
tar -rvf foo.tar test.doc
```

Los contenidos se desempaquetarán en el directorio actual.

Programación

Se deberá programar el comando *star* ("simple tar", NO "estrella"), de tal forma que acepte los comandos básicos mostrados anteriormente. Para desarrollar su programa usted debe tomar en cuenta los siguientes aspectos:

- El archivo debe organizarse internamente utilizando "memoria contigua". Esto significa que los archivos a agregar se almacenan (al principio) seguidos en el archivo empacado. Note que debe llevar control del byte en donde empieza cada archivo y el byte en donde termina.
- Al crear un archivo empacado este se crea del tamaño necesario para almacenar los archivos agregados. Cuando se borra algún contenido, el archivo empacado no cambia de tamaño sino que se lleva registro de los espacios liberados. Si posteriormente se agrega un nuevo contenido entonces se reutiliza el espacio libre. Si aún así el nuevo contenido no cabe, se hace crecer el archivo empacado. No debe utilizar ningún archivo auxiliar para hacer crecer el archivo.
- Tome en cuenta que un archivo que se agrega puede ya existir en el archivo empacado. Es decir, lo que se desea hacer es actualizar su contenido. Para esto existe la opción `update (-u)` que sobrescribirá el contenido de un archivo.
- Se debe llevar un control de los espacios libres, para asignar los huecos libres se utilizará la técnica del primer ajuste o el siguiente ajuste. Note que los espacios libres contiguos se deben fusionar.
- Este programa no utilizará los derechos de acceso, que normalmente almacenaría un archivo empacado *tar* en ambiente UNIX.
- La opción de desfragmentación (`-p`) no es estándar (no está presente en *tar*) y lo que hace es desfragmentar el contenido almacenado en el archivo empacado y liberar cualquier espacio sin utilizar. Es decir, con este comando se liberarán todos los bloques libres y el tamaño del archivo empacado se ajustará al contenido real existente. Note que no se debe utilizar un archivo temporal para realizar esta función, toda la desfragmentación se debe realizar sobre el contenido del mismo archivo.
- La opción `-v` muestra información sobre la operación ejecutada. Se puede aplicar dos veces `-vv` para ver información adicional.

```
tar -rvf foo.tar test.doc
```

Definición de estructuras de datos

Se utilizaron 3 estructuras de datos para poder implementar la memoria contigua correctamente, estas se diseñaron estudiando la materia del profesor y poder abstraer en base a los diagramas. La primera estructura de datos es la llamada file:

```
struct file {  
    char fileName[MAX_FILENAME_LENGTH];    //Nombre del archivo  
    int size;                               //Tamano del archivo  
    int startByte;                          //Byte de inicio del archivo en el directorio  
    int endByte;                            //Byte de final del archivo en el directorio  
};
```

Esta estructura de datos tiene como objetivo representar cada archivo que se guardara en tar, por lo tanto primero tendrá el nombre de cada archivo, el size representara el tamaño de cada archivo, este atributo se actualizara si se utiliza el update, también se tienen los atributos de startByte y endByte, estos guardaran donde empieza y donde termina cada archivo en el buffer de datos del archivo Tar, Necesario para casi todas las operaciones que se implementaran, estos datos se actualizaran si se utiliza el pack o update.

```
struct directory {  
    int totalSize;                          //Tamano total de todo  
    int sizeDirectory;                     //Tamano del directorio  
    int totalEntries;                      //Total de archivos en el directorio  
    struct file fileList[MAX_FILE_ENTRIES]; //Lista de los archivos en el directorio  
};
```

La segunda estructura de datos es el directory, esta estructura representa el directorio del archivo Tar, esta guarda el tamaño total del archivo Tar, cuando pesa el propio directorio, ya que es necesario para algunas operaciones, y el total de archivos que se han guardado en el archivo Tar, y una lista inicializada con las 100 entradas totales que tendrá el archivo Tar, así funcionara como un directorio y se podrá acceder a los archivos que se quiera, y ya se tiene previsto los 100 archivos máximos que tendrá.

```
// Definición de la estructura de registro de logs  
struct LogEntry {  
    char message[200];  
    struct LogEntry* next;  
};
```

La tercera estructura es los LogEntry, esta se usará para representar los mensajes que se mostraran si se utiliza el comando "--verbose", tiene un atributo message que será el mensaje, y otro atributo que es un puntero al siguiente LogEntry, como no sabemos cuál será la longitud de esta lista se utilizara una lista enlazada para más eficiencia.

Descripción detallada y explicación de los componentes principales del programa:

Mecanismo de acceso a archivos

Para el acceso a los archivos se utilizaron las funciones estándar en C para acceder y escribir archivos, las cuales son fread, fwrite, fopen, fseek, fclose y rewind, cada una de estas fue necesaria para hacer todas las operaciones de los tar. La primera sería fopen, la cual es la siguiente:

```
FILE *fopen(const char *filename, const char *mode)
```

Recibe el nombre del archivo a abrir, y el modo en que lo hará, que puede ser escritura, lectura, etc. Los modos van cambiando según la operación.

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
```

La segunda función es el fread, con fread podemos leer y guardar una porción del archivo que queramos, esta recibe el puntero donde se guardarán, cuantos bytes se guardarán, el tamaño de los datos que se guardarán, y el puntero del archivo.

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
```

La tercera función es el fwrite, con esta se podrá escribir en el archivo que queramos si lo indicamos anteriormente con fopen, esta recibe el buffer de datos que se escribirán, el tamaño de los datos que se escribirán, el número de elementos a escribir, y el puntero al archivo a escribir.

```
int fseek(FILE *stream, long int offset, int whence)
```

La cuarta función es el fseek, la cual podemos colocarlos en el punto que queramos del archivo, una función muy útil para muchas operaciones, esta recibe el puntero al archivo, en que punto nos queremos ubicar, y el modo que queramos, que existen 3, SEEK_SET, SEEK_CUR y SEEK_END.

```
void rewind(FILE *stream)
```

La quinta es rewind, la cual ubica el puntero actual al inicio del programa, recibe el puntero del archivo.

```
int fclose(FILE *stream)
```

La última es fclose que cerrara el archivo, recibe el puntero al archivo.

Estructura de directorios internos

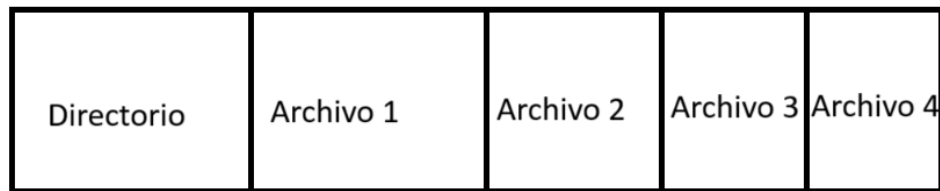
Para la estructura interna se basó en la teoría del profesor, y en los diagramas de la página web, el cual es el siguiente.

Bloques en disco

| Directorio | | |
|------------|-------|------|
| file | first | size |
| count | 0 | 4 |
| doc | 6 | 3 |
| mail | 9 | 3 |
| list | 13 | 4 |

| # | data | |
|----|------|-------|
| 0 | XXX | count |
| 1 | XXX | |
| 2 | XXX | |
| 3 | XXX | |
| 4 | - | doc |
| 5 | - | |
| 6 | YYY | |
| 7 | YYY | |
| 8 | YYY | mail |
| 9 | ZZZ | |
| 10 | ZZZ | |
| 11 | ZZZ | |
| 12 | - | list |
| 13 | TTT | |
| 14 | TTT | |
| 15 | TTT | |
| 16 | TTT | |

El directorio es casi igual, con la diferencia de que guardamos atributos adicionales como su propio tamaño o el tamaño total del archivo, este directorio será lo primero que se escribirá en el archivo, así es mucho más sencillo poder recuperar solo el directorio. A partir del directorio se escribirán los datos de los diferentes archivos, el startByte del primero es a partir del tamaño del directorio, y su endByte es el startByte más el tamaño del archivo, y así consecutivamente con cada archivo, así tenemos una lista ordenada y un buffer de datos ordenado. La lista esta siempre ordenada de menor a mayor según el startByte. Es una lista lineal de entradas de la siguiente forma.



Para esto se utilizaron las siguientes estructuras las cuales se explicaron detalladamente arriba.

```
struct file {  
    char fileName[MAX_FILENAME_LENGTH];    //Nombre del archivo  
    int size;                               //Tamaño del archivo  
    int startByte;                          //Byte de inicio del archivo en el directorio  
    int endByte;                            //Byte de final del archivo en el directorio  
};
```



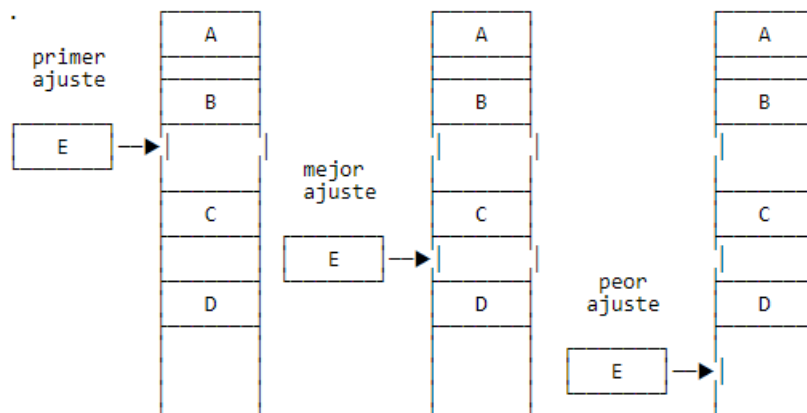
```

struct directory {
    int totalSize;           //Tamaño total de todo
    int sizeDirectory;       //Tamaño del directorio
    int totalEntries;        //Total de archivos en el directorio
    struct file fileList[MAX_FILE_ENTRIES]; //Lista de los archivos en el directorio
};

```

Estrategia de administración de espacios libres

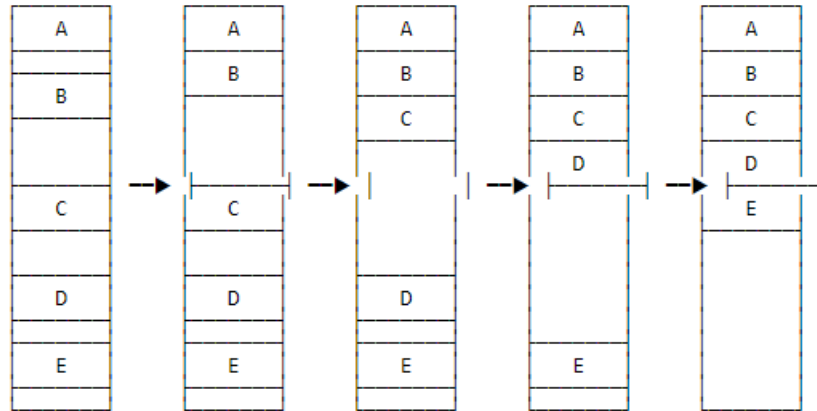
Como bien se sabe, al tratarse de un sistema de memoria contigua, crea muchos “huecos”, espacios libres de memoria que no se usan las cuales se tienen que gestionar de algún modo, para esto se utilizó una especie de estrategia de “lista enlazada”, esto consiste en que cuando se elimine un archivo, este se elimina del directorio, como es una lista que siempre esta ordenada de menor a mayor, entonces es muy fácil distinguir donde están los huecos, para esto se recorre toda la lista de archivos (por esto es una especie de lista enlazada), y se compara el endByte del archivo anterior con el startByte del archivo actual, si al restarse es un número diferente de 0, significa que existe un hueco en ese espacio, para saber de cuanto es el hueco es el propio resultado de la resta, así se puede aprovechar de muchas formas sin necesidad de una estructura adicional.



Debido a esto, para poder hacer la asignación dinámica de espacio se utilizó la estrategia de primer ajuste, esta búsqueda secuencial depende del número de archivos que están actualmente en el programa, pero la ventaja es que en el primer espacio libre que encuentre, y si es suficiente espacio entrara, sin necesidad de revisar todos, se empieza la búsqueda secuencial y se hace la resta empezando desde el tamaño del directorio al startByte del primer archivo, si la resta es diferente de 0 y el tamaño es suficiente para meter un archivo, este se meterá en esta posición, al meterse la lista de archivos se ordenara de forma que quede de menor a mayor, ya que así todo este sistema es posible. Si el espacio no es suficiente, el siguiente archivo comparara el endByte del anterior y el startByte actual para saber si existe un hueco con el tamaño suficiente, y así sucesivamente hasta recorrer toda la lista de los archivos, si no pudo encontrar un hueco suficiente, este se colocará al final. Ya que es primer ajuste.

Procedimiento de desfragmentación del archivo

Como bien se sabe, al estar eliminando e introduciendo archivos nuevos estos van cambiando la estructura y va dejando huecos, los cuales hay que eliminar al hacer desfragmentación. Para esto se utiliza la estrategia de compactación como en la siguiente imagen, solo que no tan cara como es en teoría:



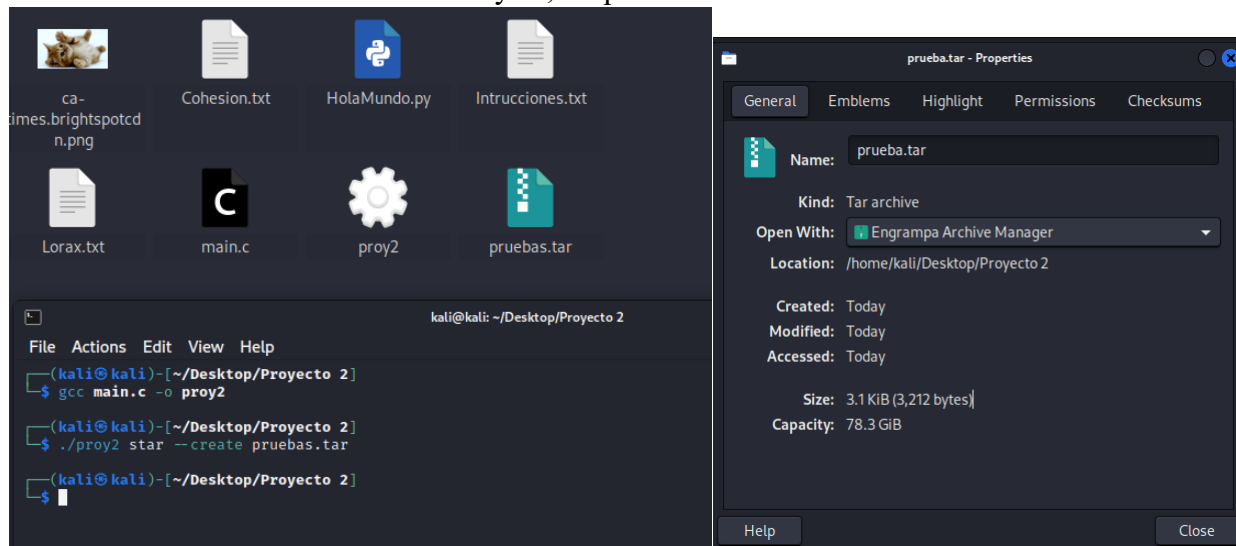
Al estar la lista de archivos ordenada de menor a mayor, para hacer la desfragmentación es tan sencillo como recorrer la lista buscando huecos como en la estrategia de primer ajuste, si este encuentra un hueco, guardara sus datos en un buffer de memoria, y escribira en tiempo de ejecución en el archivo donde estaba el hueco, y así sucesivamente con todos los archivos, así todos los huecos se van eliminando y la memoria libre se queda ubicada en el final del archivo Tar, luego de esto solamente es eliminar toda la memoria que quedo de más a partir del último archivo del directorio, así el archivo Tar se ajustara al tamaño real y todos los huecos quedaran eliminados. Gracias a que la lista siempre esta ordenada, la compactación no resulta tan cara y el método de pack resulto sencillo de implementar.

Análisis de resultados de pruebas (funcionamiento)

Se procedera a realizar pruebas que demuestren el debido funcionamiento de las diferentes opciones implementadas en el programa de sistema de archivos tar. Primero se efectuarán pruebas de las funcionalidades individuales, y después en conjunto como varios ejemplos demostrados por el profesor.

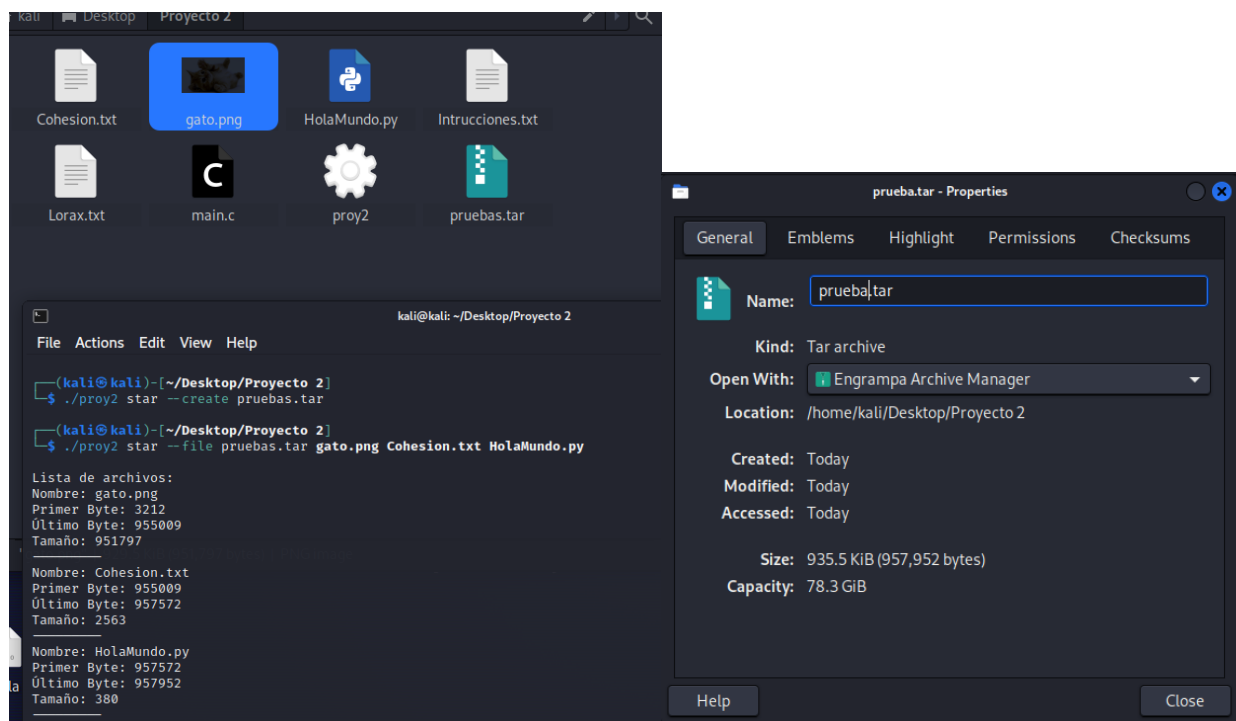
-c, --create : crea un nuevo archivo

Con el comando **--create** se le indicó crear un archivo llamado “prueba.tar”, este lo creo con un tamaño de 3212 bytes, que es el tamaño base del directorio.



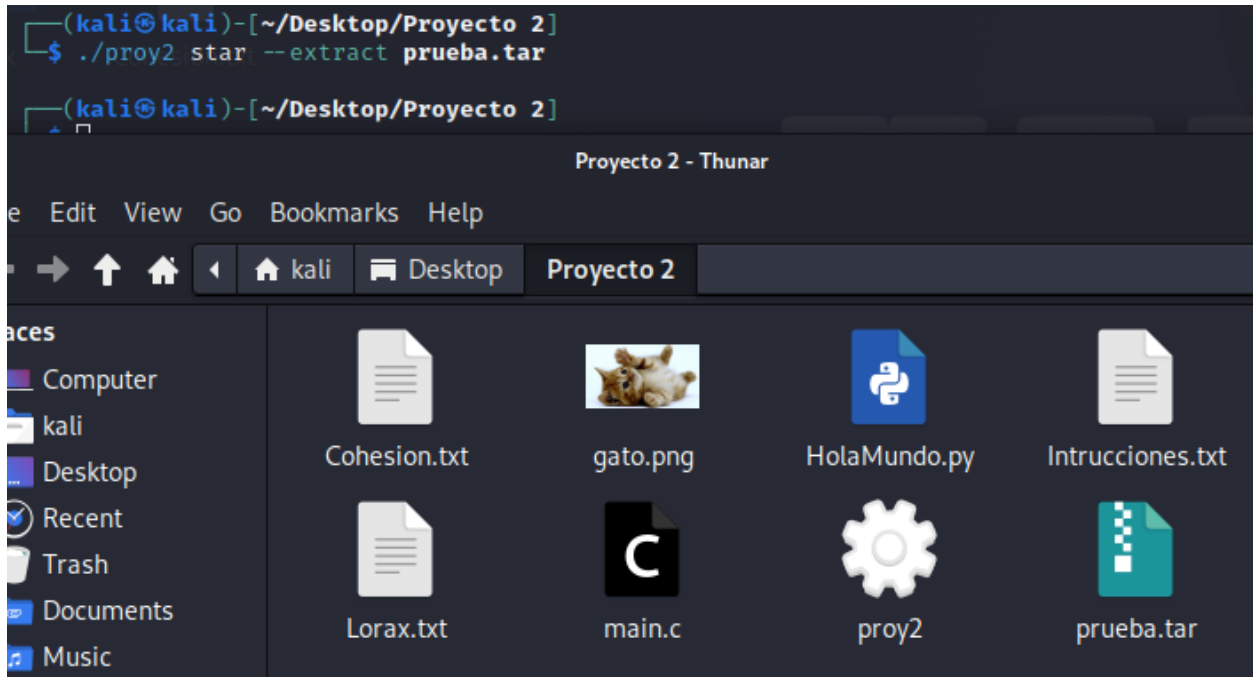
-f, --file: empaquetar contenidos de archivo, si no está presente asume la entrada estándar.

El comando **--file** empaquetó en el .tar, 3 archivos diferentes, una imagen (gato.png), HolaMundo.py y Cohesion.txt. Se puede observar los archivos en el directorio, el byte de inicio y de final. Además, también el cambio de tamaño en el tar.



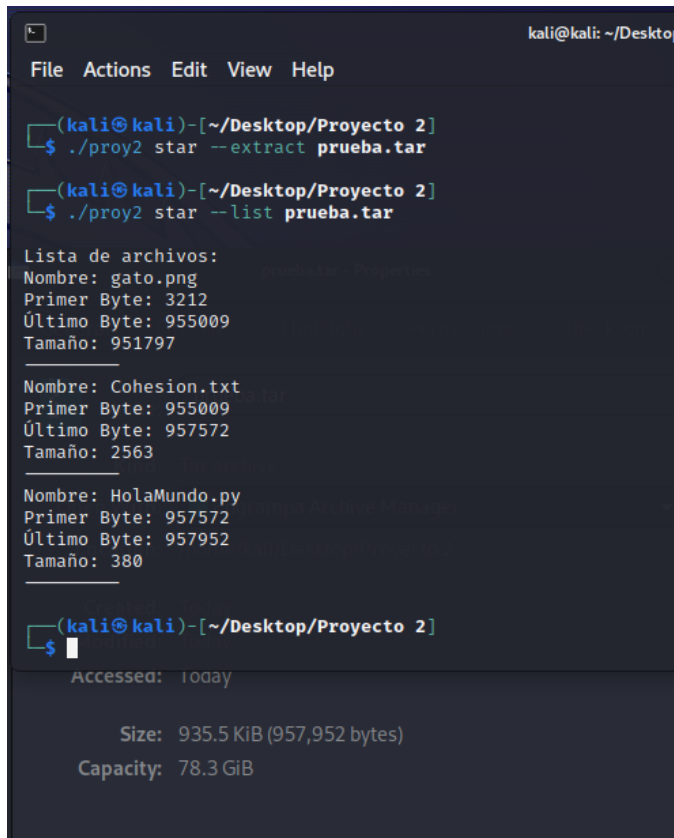
-x, --extract : extraer de un archivo

El **--extract** extrae del archivo tar los 3 archivos anteriormente empaquetados. Se puede observar que lo realiza de forma correcta. Antes de realizar esta prueba estos archivos fueron removidos de la carpeta.



-t, --list: listar los contenidos de un archivo

El comando **--list**, lista los archivos en el directorio con sus especificaciones.



```
kali@kali: ~/Desktop/Proyecto 2
File Actions Edit View Help

(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --extract prueba.tar

(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --list prueba.tar

Lista de archivos:
Nombre: gato.png
Primer Byte: 3212
Último Byte: 955009
Tamaño: 951797

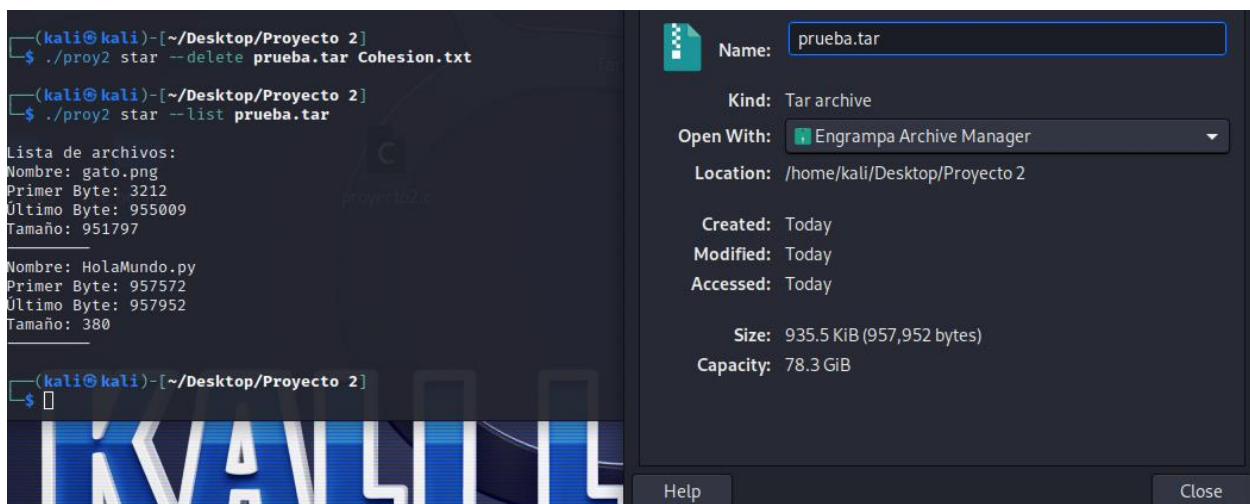
Nombre: Cohesion.txt
Primer Byte: 955009
Último Byte: 957572
Tamaño: 2563

Nombre: HolaMundo.py
Primer Byte: 957572
Último Byte: 957952
Tamaño: 380

(kali@kali)-[~/Desktop/Proyecto 2]
$
```

--delete: borrar desde un archivo

Se procederá a borrar el archivo del medio con el comando **--delete**, Cohesion.txt. Esto para dejar un hueco en el medio donde se podrá demostrar el funcionamiento de los siguientes comandos. Se puede observar que el archivo fue borrado al listar los que pertenecen al directorio, pero el tamaño del archivo no cambia. Lo que deja un bloque vacío en el medio de 2563 bytes.



```
(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --delete prueba.tar Cohesion.txt

(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --list prueba.tar

Lista de archivos:
Nombre: gato.png
Primer Byte: 3212
Último Byte: 955009
Tamaño: 951797

Nombre: HolaMundo.py
Primer Byte: 957572
Último Byte: 957952
Tamaño: 380

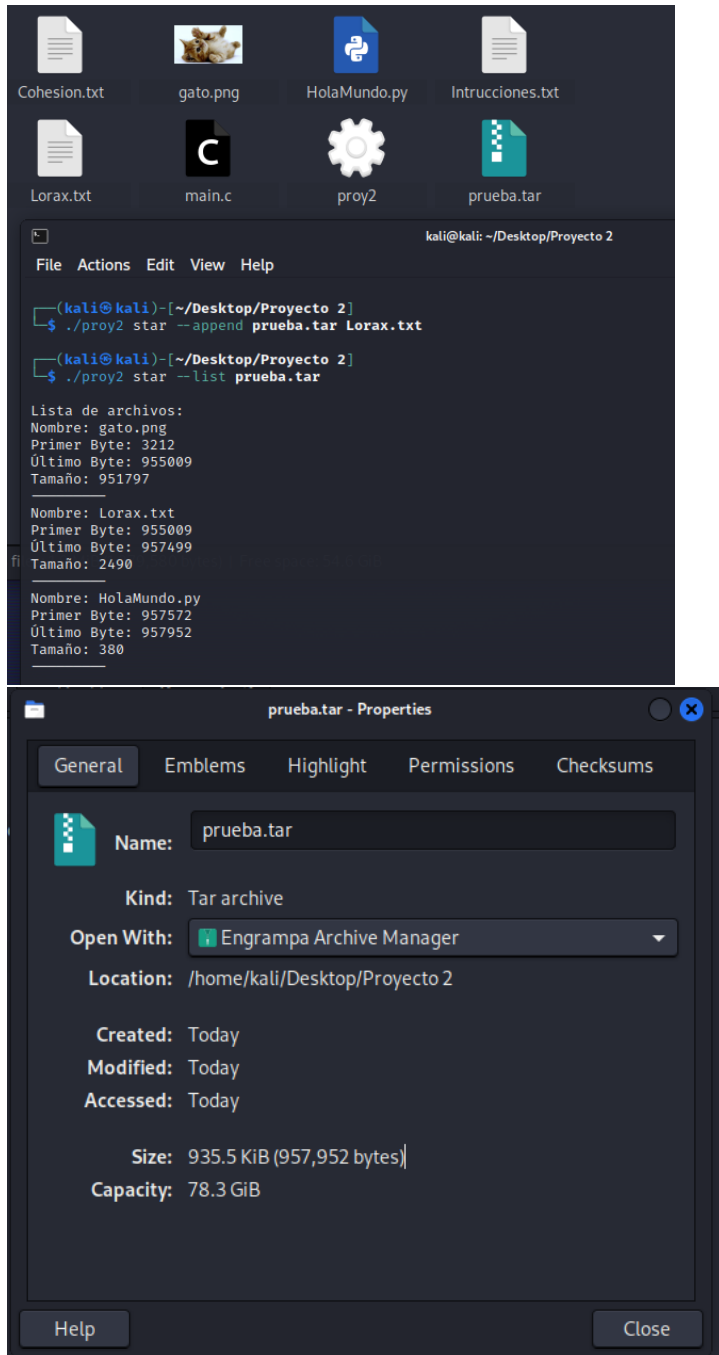
(kali@kali)-[~/Desktop/Proyecto 2]
$
```

File Manager Details:

- Name: prueba.tar
- Kind: Tar archive
- Open With: Engrampa Archive Manager
- Location: /home/kali/Desktop/Proyecto 2
- Created: Today
- Modified: Today
- Accessed: Today
- Size: 935.5 KiB (957,952 bytes)
- Capacity: 78.3 GiB

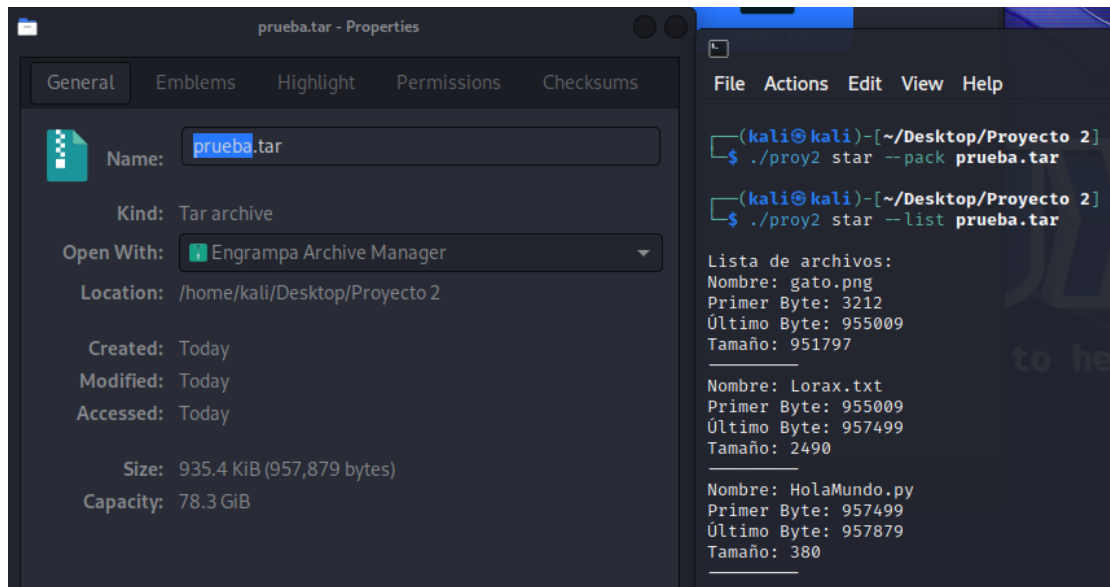
-r, --append: agrega contenido a un archivo

Se realiza un **--append** en el archivo, del archivo Lorax.txt que justo pesa 2490 bytes, por lo que se debería insertar en el bloque vacío, dejando un pequeño espacio de 73 bytes. El archivo tampoco debería cambiar de tamaño.



-p, --pack: desfragmenta el contenido del archivo (no presente en tar)

Se le procede a hacer **--pack** al archivo por lo que elimina los 73 bytes y el bloque vacío. Además, se debe volver actualizar el tamaño del archivo tar, reescribir los archivos y actualizar los bytes de iniciación y finalización. Se puede ver como cambiaron los bytes de inicio y una diferencia en el tamaño de 73 bytes respecto con el anterior.



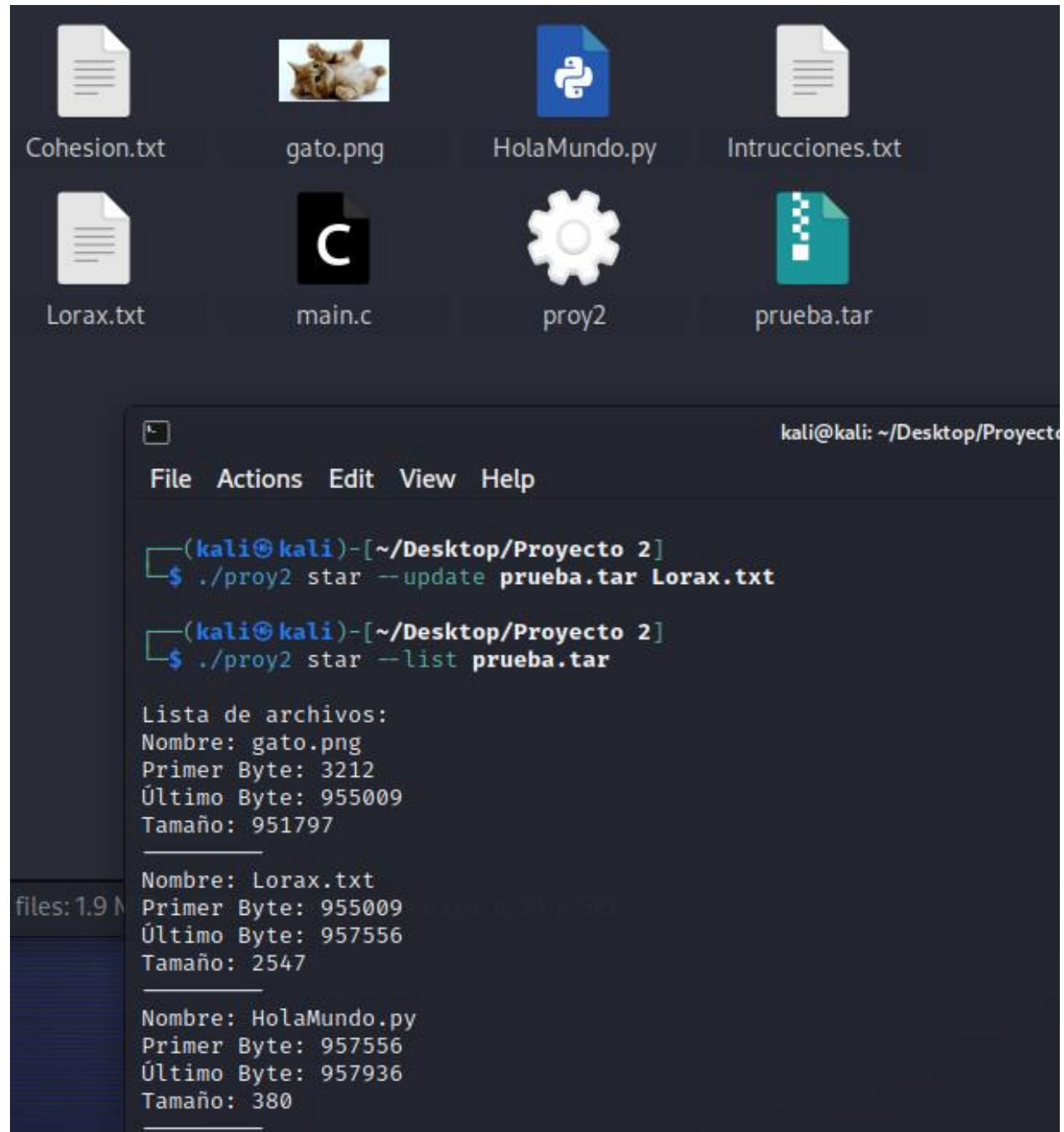
-v, --verbose: ver un reporte de las acciones a medida que se van realizando

Probar el **--verbose** solo no tiene mucho sentido ya que este no ha realizado pasos anteriores. Por lo que no guarda muchos logs para enseñar. Las pruebas compuestas más adelante se podrá observar un mejor funcionamiento de esta función.

```
(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --verbose prueba.tar
Registros de logs:
Informacion leida del archivo:
prueba.tar
```

-u, --update: actualiza el contenido del archivo

El comando **--update** actualiza el contenido de un archivo. Se modificará el archivo Lorax.txt donde se le agregaran 57 bytes extra a su contenido y luego actualizarlo en el tar. Se deben actualizar los bytes de inicio y posteriores a este archivo. Además, al archivo se le agregan los 57 bytes extra porque al realizar un pack antes de ejecutar esta función, se eliminan los espacios en blanco.



The screenshot shows a Kali Linux desktop environment. At the top, there is a file manager window displaying a grid of files: Cohesion.txt, gato.png, HolaMundo.py, Intrucciones.txt, Lorax.txt, main.c, proy2, and prueba.tar. Below the file manager, a terminal window is open, showing the execution of the 'star' command to update the 'prueba.tar' archive with the contents of 'Lorax.txt'.

```
kali@kali: ~/Desktop/Proyecto 2
File Actions Edit View Help

(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --update prueba.tar Lorax.txt

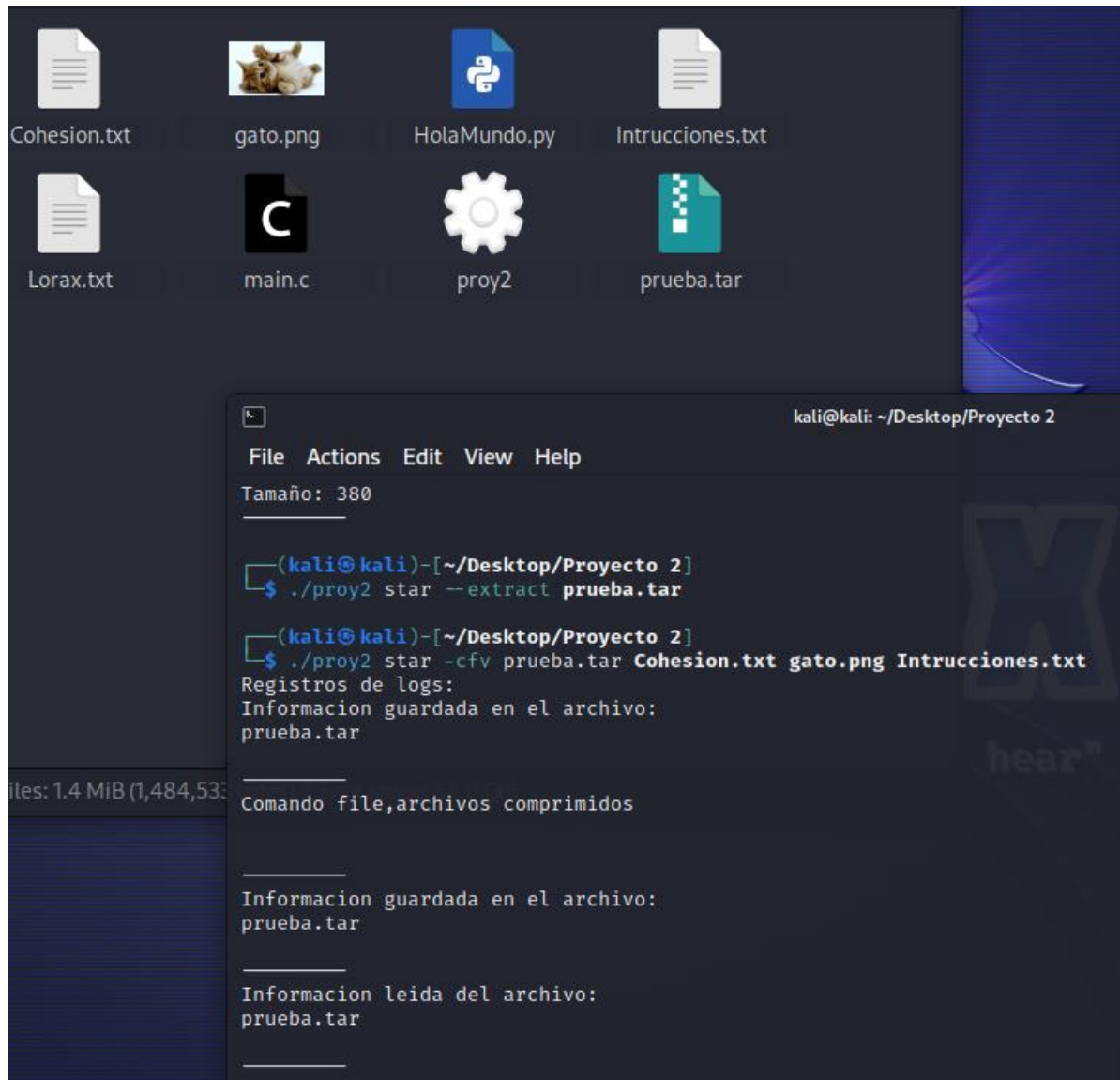
(kali@kali)-[~/Desktop/Proyecto 2]
$ ./proy2 star --list prueba.tar

Lista de archivos:
Nombre: gato.png
Primer Byte: 3212
Último Byte: 955009
Tamaño: 951797
-----
Nombre: Lorax.txt
Primer Byte: 955009
Último Byte: 957556
Tamaño: 2547
-----
Nombre: HolaMundo.py
Primer Byte: 957556
Último Byte: 957936
Tamaño: 380
-----
```


Pruebas en conjunto

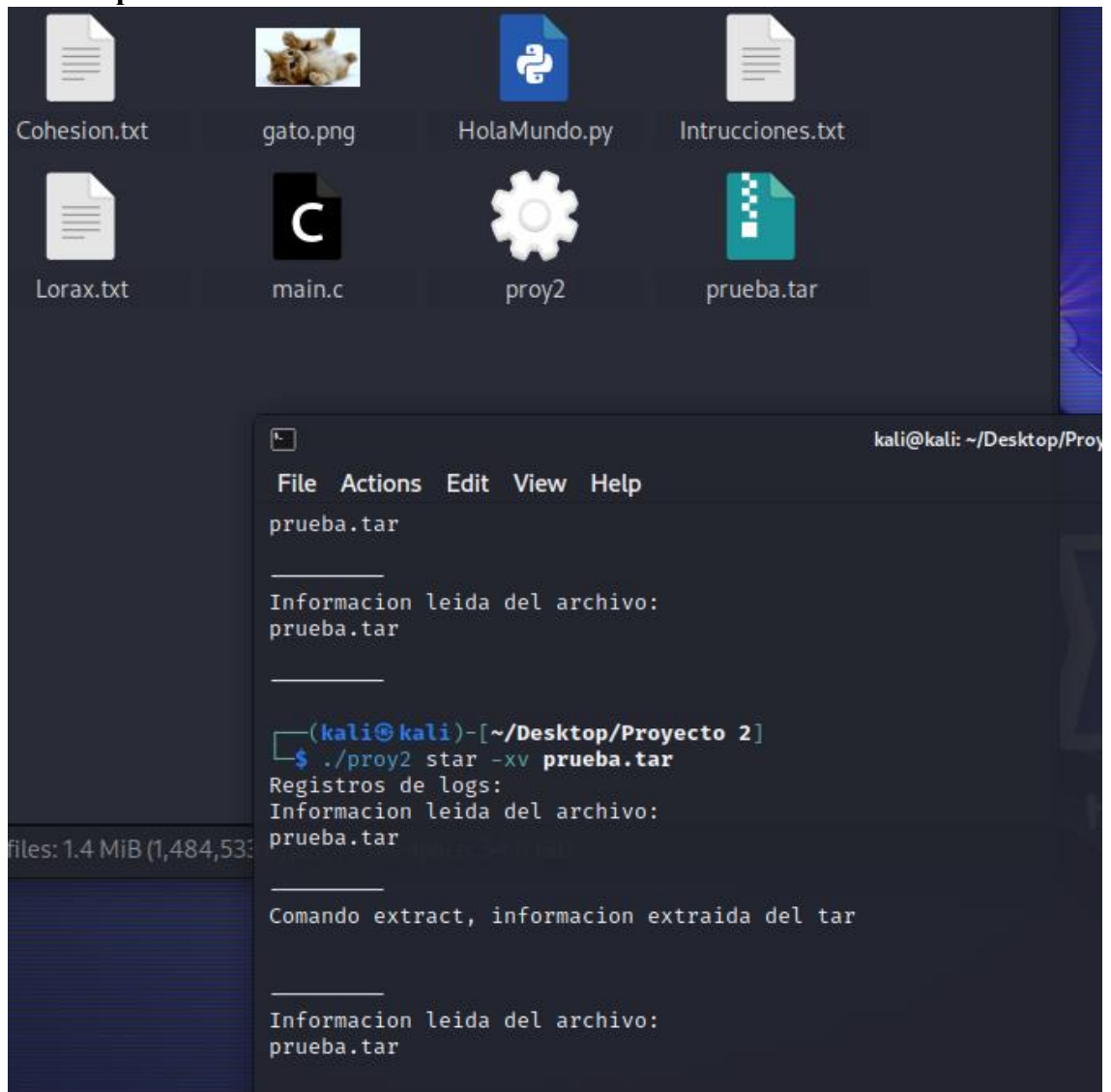
Empacar varios archivos llamados "Cohesion.txt", "gato.png", "Instrucciones.txt" y guardar los datos en "prueba.tar", lo haríamos con la instrucción y ver los logs sencillos.

- **star -cfv prueba.tar Cohesion.txt gato.png Instrucciones.txt**



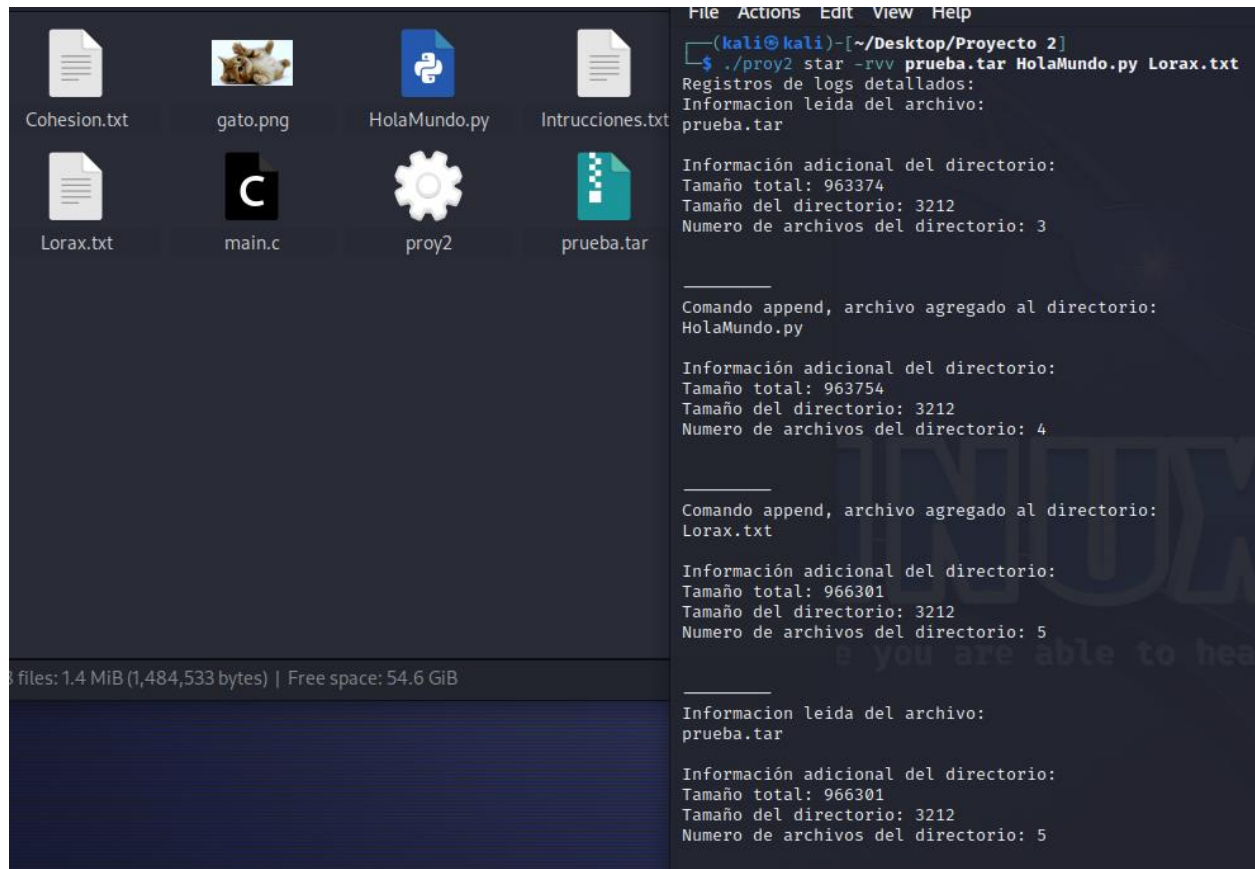
Si queremos desempaquetar todo el contenido de un archivo llamado en "prueba.tar", podemos utilizar un comando como este y ver los logs sencillos.

- **star -xv prueba.tar**



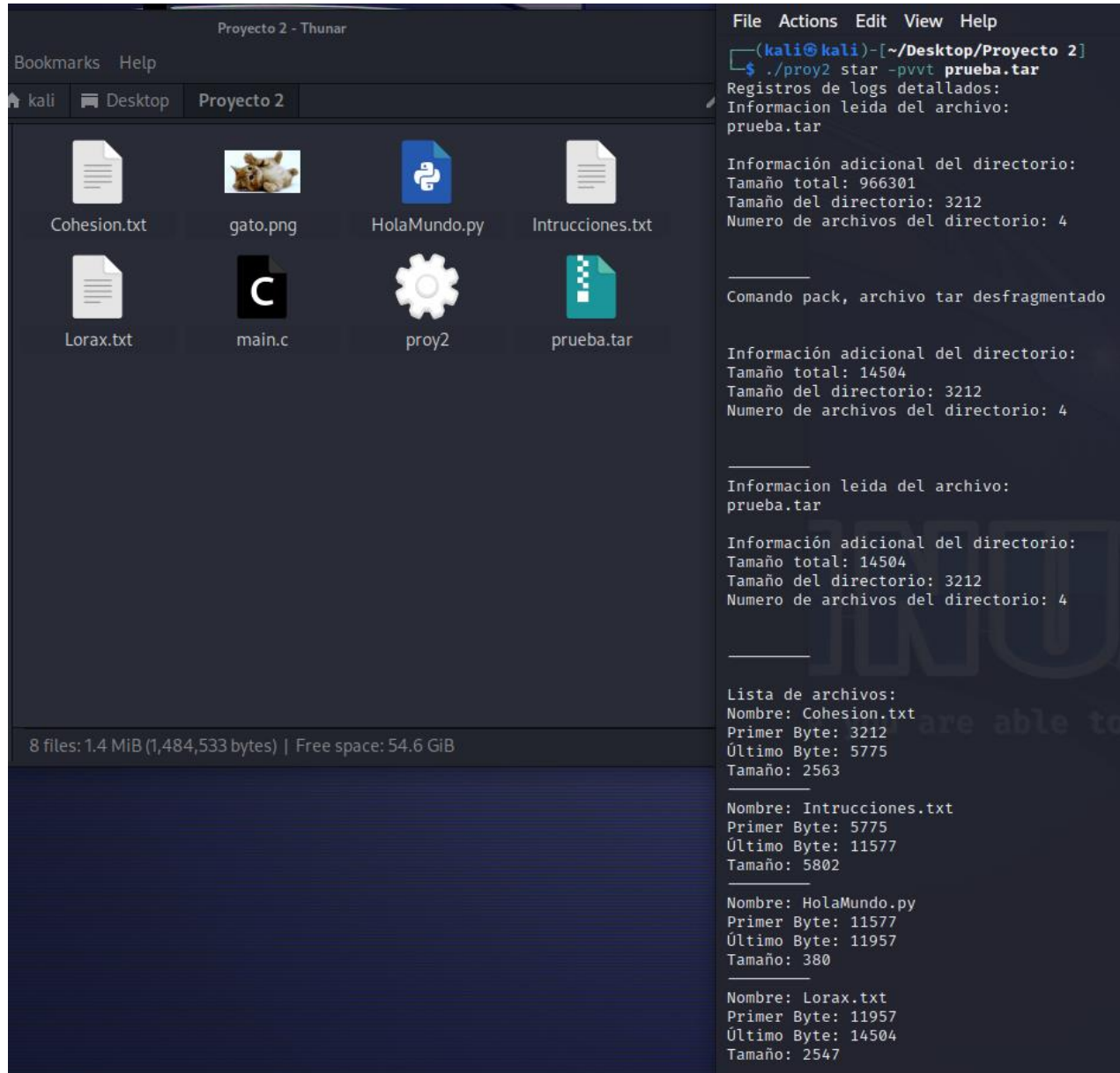
Para agregar "HolaMundo.py" y "Lorax.txt" a "prueba.tar" y ver la información adicional, se ejecutaría

- **star -rvv prueba.tar HolaMundo.py Lorax.txt**



Por último, después de hacer un `--delete` de "gato.png", desfragmentar el tar, ver los logs detallados y por último listar los archivos.

- `star --delete prueba.tar gato.png`
- `star -pvvt prueba.tar`



Conclusiones

- Los archivos Tar son una gran manera de comprimir archivos para poder transportarlos o guardar lo que queramos fácilmente.
- Se debe tener muy pensada la estructura antes de empezar a programar, si no puede haber muchos problemas a la hora de desarrollar las funciones si no se pensaron bien.
- Así como las estructuras, también se debe tener bien pensada las estrategias a la hora de gestionar el espacio libre, para que así la desfragmentación sea más sencilla.
- Tener muy bien en claro el startByte y endByte de cada archivo, si no algunas funciones como extract no funcionarían correctamente.
- Siempre probar combinación de funciones, ya que pueden dar resultados inesperados que es mejor probarlos antes.
- Seguir la teoría del profesor para poder crear buenas estrategias de gestión del espacio libre o desfragmentación.
- Informarse sobre las funciones de fread, fseek, fwrite, etc... es útil ya que puede ayudar a crear muchas funciones.
- Comentar bien el programa creado ya que al ser códigos tan extenso y complejos puede hacer perder incluso al mismo creador.