



Universidad Nacional Autónoma de México Facultad de Ingeniería



Semestre:
2025-2

Materia:
Sistemas Operativos

Grupo:
06

Profesor:
Gunnar Eyal Wolf Iszaevich

Integrantes:
Avila Reyes Iker
Romero Pizano Christian Gustavo

Tarea:
Documento de Presentación

Fecha de Entrega:
02/05/25

BTRFS

¿Qué es el BTRFS?

BTRFS (“B-Tree File System” o “Sistema de Archivos de Árbol B”) es un sistema de archivos en Linux diseñado para ser robusto, eficiente y escalable. Entre sus herramientas principales se incluyen sistemas incorporados de compresión, control de versiones y detección de errores.^{[2][3][4]} Está basado en la técnica Copy-On-Write, lo que le da un uso de almacenamiento más eficiente. El sistema utiliza una estructura de datos de “árboles B”^[5] (de ahí su nombre), y está diseñado para poder manejar distintos usos y cargas de trabajo, por lo que se continúa actualizando constantemente para mantener un rendimiento decente.^[1]

BTRFS es relativamente joven. Fue lanzado en 2007 y declarado estable hasta el 2013.

¿Qué es un sistema de archivos?

Un sistema de archivos es una estructura utilizada por un sistema operativo para organizar y administrar archivos en un dispositivo de almacenamiento, como pueden ser un disco duro, una unidad de estado sólido (SSD) o un dispositivo flash USB. Este sistema define cómo se almacenan, acceden y organizan los datos en el dispositivo de almacenamiento.^[13]

Por decir nombres, los sistemas de archivos más comunes son:

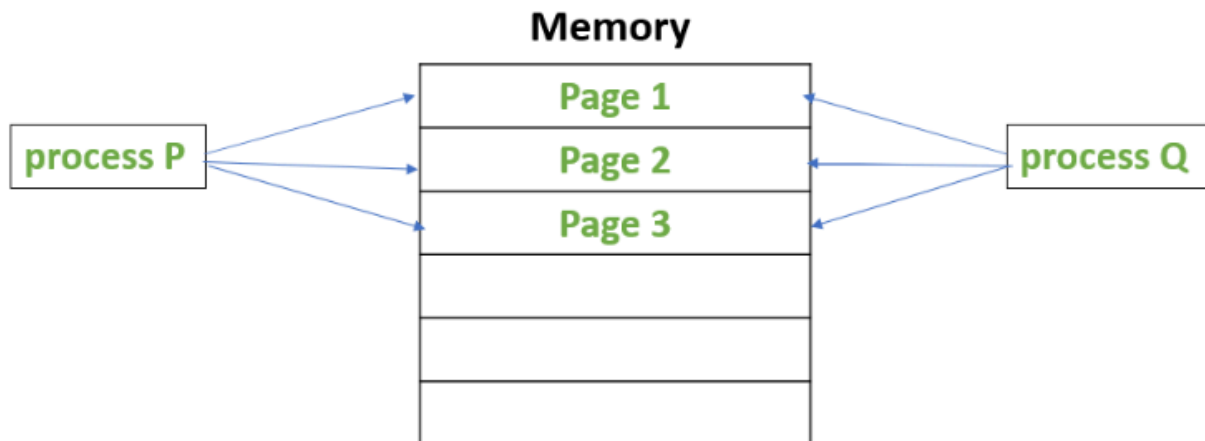
- FAT y exFAT: Tabla de Asignación de Archivos y su versión extendida.
- NTFS: Sistema de Archivos de Nueva Tecnología predeterminado de Windows.
- APFS: Sistema de Archivos de Apple.
- HFS, HFS+: Sistema de Archivos Jerárquico.
- Ext4: Cuarto Sistema de Archivos Extendido, ampliamente utilizado en Linux.

Copy-On-Write.

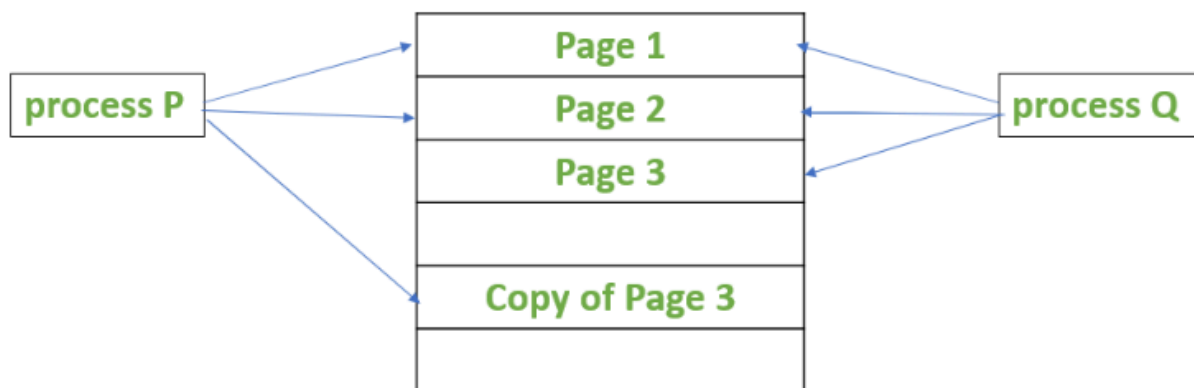
Mencionamos que el sistema está basado en la técnica Copy-On-Write. Pero, ¿qué significa esto?

Copy-On-Write, también conocido como “Shadowing”^[6] o “Implicit Sharing”^[7], es una técnica que busca maximizar el uso eficiente de recursos y minimizar el copiado innecesario de datos. Básicamente, comparte bloques de almacenamiento entre archivos hasta que uno necesita ser modificado. Establece que en casos de lectura, los

datos se pueden compartir sin problema por varios archivos, y solo es necesario copiarlos cuando alguno intenta escribir sobre ellos. En éstos casos, la copia se guarda en una nueva dirección de memoria sin sobrescribir los datos originales. Como consecuencia, se mejora la integridad de los datos y se facilita la creación de snapshots.



Lectura compartida por dos procesos.

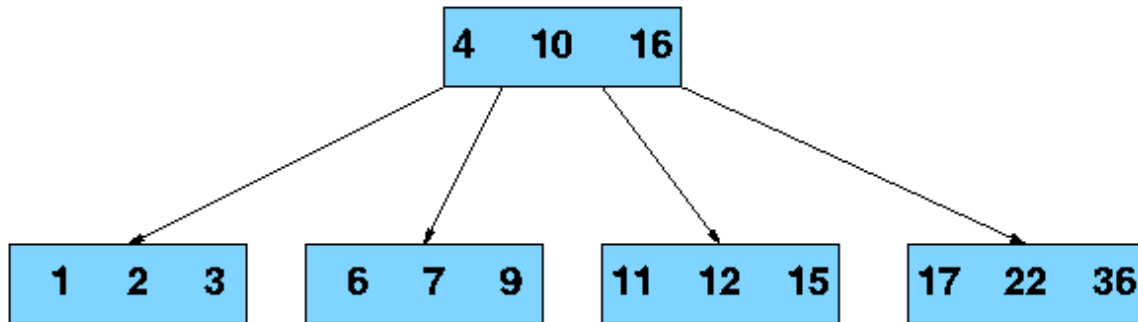


Proceso P realiza escritura en la página 3.^[15]

Árboles B.

Otra cosa que se mencionó es que BTRFS utiliza una estructura de árboles B. Los árboles B son una estructura de datos cuyo objetivo es administrar grandes cantidades de datos, manteniéndolos organizados y fácilmente accesibles para minimizar la cantidad de accesos al disco, razón por la cual son comúnmente utilizados dentro de los sistemas de archivos.^{[5][8]}

A diferencia de los árboles binarios tradicionales, un árbol B de orden 'm' puede tener múltiples claves y ramificaciones, permitiendo que cada nodo almacene hasta 'm-1' claves y 'm' hijos. Esta característica reduce significativamente la altura del árbol, minimizando el número de accesos al disco y mejorando el rendimiento en las operaciones fundamentales del árbol: inserción, eliminación y búsqueda.^[14]



Ejemplo: Árbol con 3 claves y 4 hijos en el nodo raíz.

Reglas y Características de un Árbol B.

Una característica importante es que todos sus nodos hojas se mantienen en el mismo nivel, garantizando un equilibrio automático, y para agilizar las consultas, las claves dentro de cada nodo se almacenan en un orden ascendente. Además, un árbol B de tamaño 'm' debe de seguir las siguientes reglas:

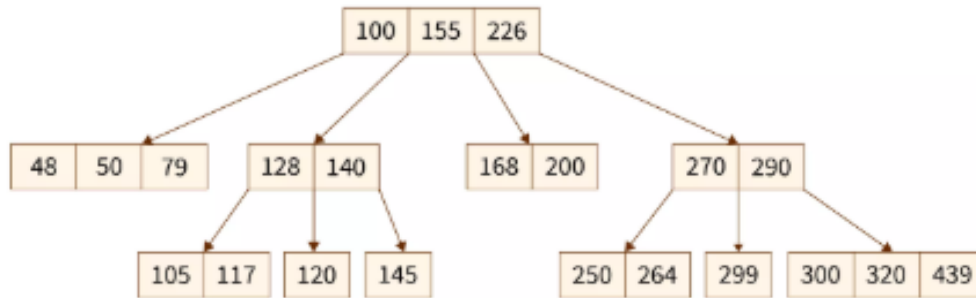
- A excepción de la raíz, todo nodo interno (no hoja) debe tener al menos $\lceil m/2 \rceil$ hijos.
- A excepción de la raíz, todo nodo debe contener al menos $\lceil m/2 \rceil - 1$ claves.
- Si la raíz es hoja (el único nodo del árbol) puede tener entre 1 y $\lceil m-1 \rceil$ claves.
- Si la raíz no es una hoja, debe tener al menos 2 hijos y 1 clave.
- Si un nodo excede el número de claves $\lceil m-1 \rceil$ se divide.
- Si un nodo queda por debajo del mínimo de claves $\lceil m/2 \rceil - 1$ se fusiona o redistribuye con uno de sus nodos hermanos.

Por ejemplo, para un árbol B de orden 5 tenemos:

Máximo por nodo: 4 claves y 5 hijos.

Mínimo por nodo (no raíz): 2 claves y 3 hijos.

Raíz (si no es hoja): Al menos 1 clave y 2 hijos.



Arbol B de orden 4.

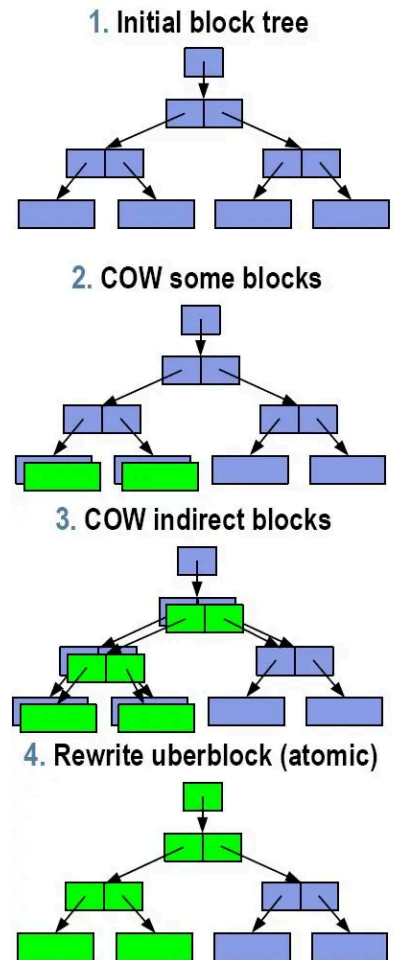
¿Cómo es el proceso de escritura?

Digamos que queremos modificar un archivo llamado "archivo.txt". Lo primero que se realiza es la identificación de los bloques de datos. Siguiendo el sistema de árbol B, cada nodo contiene referencias a bloques de datos (por ejemplo, bloques en el disco donde está guardado el archivo). Para encontrar archivo.txt, recorre desde el Árbol Raíz → Árbol de Metadatos → Entrada del archivo.

Una vez encontrado, como se quiere modificar el archivo, se realiza una copia. En lugar de sobrescribir el archivo sobre ese bloque, se asigna un nuevo bloque de memoria, y se escribe la nueva versión de los datos en ese bloque.

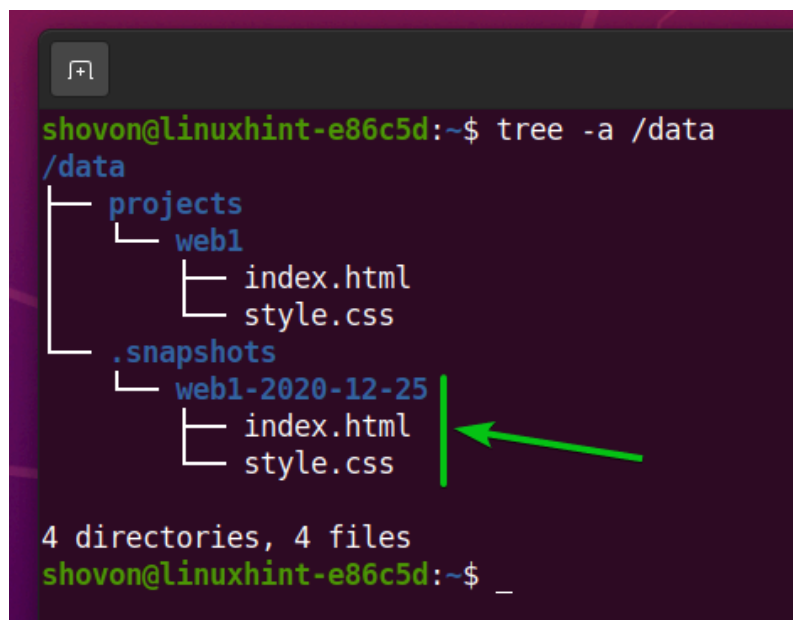
Finalmente, se actualizan los nodos del árbol B. Primero, se copia el nodo padre del bloque de datos antiguo a un nuevo nodo y éste se actualiza para que apunte al nuevo bloque. Este proceso se repite recursivamente hacia arriba hasta llegar a la raíz del árbol, la cual se copia y modifica para que apunte a la nueva versión del árbol.^[16]

Al mantener la versión anterior del archivo, se puede regresar a ella en caso de corrupción o algún otro fallo.



Snapshots.

También se mencionó que la técnica de Copy-On-Write facilita la creación de snapshots debido a que nunca se sobrescriben los datos originales. Un snapshot es básicamente un subvolumen de datos que mantiene referencias a versiones anteriores de su estructura. Como se vió en el ejemplo anterior, la versión original del archivo no se elimina, y se mantiene una referencia hacia ella. Es precisamente esta referencia a la versión original un snapshot. Una imagen del estado del sistema de archivos en cierto punto del tiempo.^{[2][9]}



```
shovon@linuxhint-e86c5d:~$ tree -a /data
/data
├── projects
│   └── web1
│       ├── index.html
│       └── style.css
└── .snapshots
    └── web1-2020-12-25
        ├── index.html
        └── style.css

4 directories, 4 files
shovon@linuxhint-e86c5d:~$ _
```

A terminal window with a dark purple background. It shows the output of the command 'tree -a /data'. The output is a tree structure. The root is '/data'. It has two main branches: 'projects' and '.snapshots'. 'projects' contains a subdirectory 'web1' which has two files: 'index.html' and 'style.css'. '.snapshots' contains a subdirectory 'web1-2020-12-25' which also has two files: 'index.html' and 'style.css'. At the bottom, it says '4 directories, 4 files'. A green arrow points to the 'web1-2020-12-25' directory name.

Ejemplo de un Snapshot.^[17]

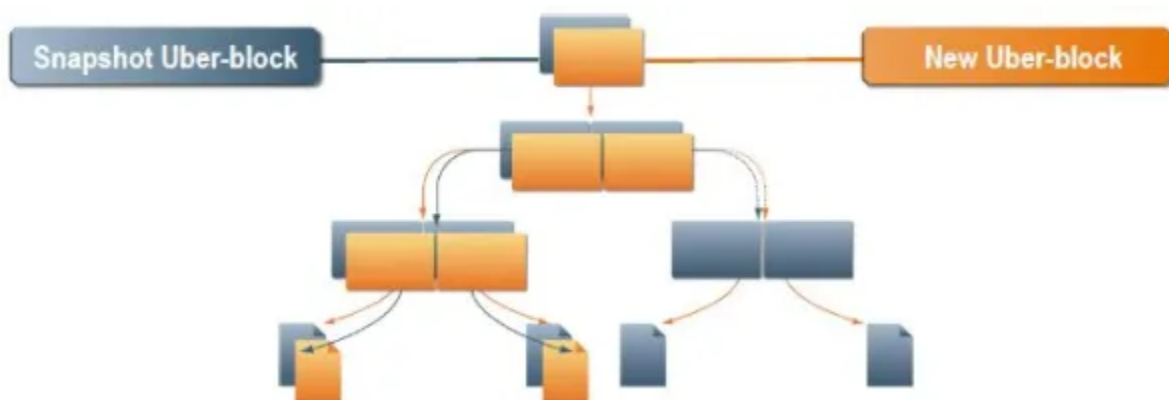
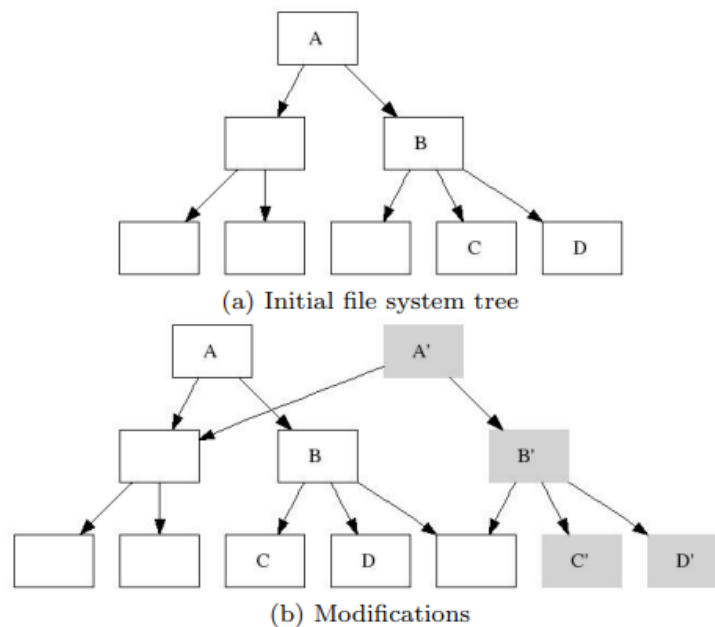
¿Cómo funcionan?

Utilicemos un nuevo ejemplo. Imaginemos que tenemos un archivo “holamundo.txt” que contiene el texto “Hola, mundo.”, y decidimos crear un snapshot en la carpeta “snapshots/holamundo-v1.txt”. Entramos a esta carpeta, abrimos el archivo de texto, y vemos que contiene el mismo “Hola, mundo.”.

Lo interesante comienza cuando decidimos modificar nuestro archivo. Digamos que modificamos el “holamundo.txt” original para que diga: “Hola, mundo. Soy un archivo de texto.”. ¿Qué sucede en el archivo modificado? Básicamente, en vez de copiar completamente la cadena de “Hola, mundo.” que ya existe en el snapshot que creamos, el archivo está haciendo referencia a ese “Hola, mundo.” del snapshot, y

simplemente hace las modificaciones necesarias en base a eso (añadir “Soy un archivo de texto.” al final).

¿De qué nos sirve esto? Ahora tomemos como ejemplo un archivo de 5 GB, del cual queremos otra versión con tan solo unos pequeños detalles diferentes. En vez de copiar el archivo completamente y tener dos archivos que llegarían a los 10 GB, simplemente se hace referencia al archivo original y de ahí se realizan las modificaciones necesarias.^[9] A esto nos referíamos cuando mencionamos que BTRFS hace un uso más eficiente de los recursos.



Diagramas de un snapshot.^{[6][16]}

Otros Usos de los Snapshots.

Además del uso eficiente del almacenamiento, los snapshots son fundamentales en otros entornos donde la integridad y la disponibilidad de los datos son una prioridad. Dos casos de uso en donde destacan estas condiciones son:

1. RespalDOS seguros en servidores.

Para evitar inconsistencias dentro de un respaldo (por ejemplo, si un archivo se modifica mientras se copia) se lleva a cabo la siguiente secuencia:

- 1.- Se congela el estado del sistema: Se crea un snapshot del sistema de archivos.
- 2.- Se respalda el snapshot: Se copian los datos del snapshot en un medio de almacenamiento externo.
- 3.- Se elimina el snapshot: Si se buscan liberar recursos es posible borrar el snapshot.

2. Actualizaciones reversibles.

Antes de actualizar algún sistema operativo o software crítico, un snapshot permite revertir los cambios en caso de que algo falle.

Detección de Errores.

Aparte de la técnica de Copy-On-Write y el uso de Snapshots, otra manera en la que BTRFS protege la integridad de los datos es a través de las sumas de verificación, o “checksums”. Cada vez que se escribe un bloque en el disco, BTRFS realiza un checksum para ese bloque. Este checksum se guarda en un árbol de checksums separado. Así, cada vez que se lee un bloque, se vuelve a calcular su checksum y se compara con el almacenado para verificar que no haya corrupción.^[2]

Volviendo al ejemplo inicial, tenemos nuestro “archivo.txt”, y lo modificamos. Digamos que el archivo modificado se va a guardar en el bloque de código “B1”. Antes de escribir nuestro archivo en el disco, se calcula su checksum con alguno de los varios algoritmos disponibles, y esto nos da como resultado “C1”. BTRFS entonces guarda “C1” en el árbol de checksums, y este guarda la dirección lógica del bloque (en este caso, “B1”).

Ahora, cuando intentamos leer “archivo.txt”, BTRFS lo localiza en “B1”, y realiza nuevamente el cálculo del checksum. Digamos que esto nos resulta en “C2”, y comienza a buscar en el árbol de checksums la entrada para “B1”. Como estaba en “C1”, y en el nuevo cálculo del checksum obtuvimos “C2”, esto significa que hubo algún

tipo de corrupción. Si hubiera dado “C1” otra vez, esto hubiera significado que no se encontraron problemas.

Seguridad y fiabilidad

El BTRFS también puede integrarse dentro de sistemas de seguridad (como SELinux o AppArmor) lo que lo vuelve una opción aún más confiable en entornos donde se requiere un control de acceso más riguroso. Esto permite establecer políticas de acceso más estrictas y proteger los datos contra programas maliciosos o no autorizados^[2]. Esta integración es particularmente útil en entornos con contenedores o múltiples usuarios, donde se desea mantener una separación clara y segura entre distintos espacios de trabajo.

Por otro lado, es importante mencionar algunas limitaciones con las que se cuenta. Aunque muchas de sus funciones son estables, todavía existen ciertas características que se podrían considerar como “experimentales”, tales como los módulos RAID 5 y 6. Estas versiones podrían presentar algunos problemas de integridad en algunos entornos o en equipos que presenten discos defectuosos o un kernel desactualizado si no se toman las debidas precauciones.

A pesar de estas advertencias, este sigue siendo uno de los sistemas de archivos más robustos, el cual es capaz de ofrecer una capa sólida de seguridad para cualquier administrador de sistemas siempre y cuando se cuenten con las herramientas adecuadas de monitoreo y mantenimiento.

Herramientas de gestión.

Este sistema de archivos cuenta con una serie de herramientas diseñadas para facilitar su administración. Una de las más utilizadas es su conjunto de comandos, el cual permite realizar tareas como la creación de subvolumenes, balanceo de datos y verificación de errores entre otras cosas. Los comandos más comunes suelen ser:

- *btrfs subvolume create* para crear subvolumenes independientes ^[10]
- *btrfs snapshot* para generar snapshots manuales
- *btrfs scrub start* para iniciar un proceso de verificación de datos y checksums en segundo plano ^[11]
- *btrfs balance start* para reorganizar los datos en el sistema y optimizar el uso del espacio
- *btrfs check* para inspeccionar y reparar el sistema de archivos

Comparativas y aplicaciones.

El BTRFS se suele comparar con otros sistemas de archivos populares en Linux, tales como EXT4, XFS y ZFS. En comparación con el primero, BTRFS nos ofrece funciones mucho más avanzadas como lo son las snapshots y la detección automática de errores, aunque EXT4 sigue siendo más estable en sistemas de propósito general y con menos necesidades de administración avanzada^[12].

Respecto a ZFS, ambos buscan garantizar la integridad de los datos y ofrecer snapshots, compresión y gestión avanzada de volumen. Sin embargo, el BTRFS se integra en el kernel de Linux de una manera más fácil, mientras que ZFS requiere de módulos externos ofreciendo funciones más maduras, especialmente en la recuperación ante fallos severos^[12].

En cuanto a sus aplicaciones, BTRFS es ampliamente utilizado en entornos donde la eficiencia, la integridad de datos y la facilidad de mantenimiento son clave. Algunas distribuciones como Fedora y Ubuntu (en servidores) lo integran como opción por defecto o recomendada. También es una opción popular en plataformas de virtualización o contenedores, donde los snapshots ofrecen una enorme ventaja en términos de backups, restauraciones y despliegue rápido de entornos.

Conclusión.

En conclusión, BTRFS es un sistema de archivos altamente eficiente gracias a su diseño y diversos algoritmos. La filosofía de Copy-On-Write junto con el uso de árboles B, evita casos de corrupción y nos ahorra valioso tiempo y memoria cuando se tienen varios procesos que solo leen datos sin realizar modificaciones. Además, facilita la creación de snapshots, los cuales nos permiten alcanzar un mejor uso de la memoria del equipo, evitando copiar datos redundantes cuando simplemente se puede hacer referencia a los datos originales en archivos modificados. Otro uso de los snapshots es facilitar la gestión y control de versiones, que es bastante útil para la realización de backups y restauraciones de sistema. Finalmente, la detección de errores añade todavía otra capa de seguridad para mantener la integridad de los datos y detectar posibles problemas en el sistema. Estas son tan solo algunas de las herramientas con las que cuenta BTRFS, pero existen muchas otras que lo convierten en un sistema para manejar archivos sumamente robusto.

Bibliografia.

1. Rodeh, O., Bacik, J., & Mason, C. (2013). BTRFS: The Linux B-tree filesystem. ACM Transactions on Storage (TOS). <https://doi.org/10.1145/2501620.2501623>.
2. The Btrfs Wiki. (s.f.). Btrfs Wiki. <https://btrfs.readthedocs.io>.
3. The Linux Kernel Archives. (n.d.). Btrfs - Linux Kernel Documentation. <https://www.kernel.org/doc/html/latest/filesystems/btrfs.html>.
4. Red Hat. (s.f.). Chapter 4. Btrfs. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/storage_administration_guide/ch-btrfs#ch-btrfs.
5. Bayer, Rudolf. (2008). B-tree and UB-tree. Scholarpedia. http://www.scholarpedia.org/article/B-tree_and_UB-tree.
6. Rodeh, Ohad. (2007). ACM Transactions on Computational Logic. IBM. <https://web.archive.org/web/20170102212904/http://liw.fi/larch/ohad-btrees-shadowing-clones.pdf>.
7. Qt group. (s.f.). Implicit Sharing. <https://doc.qt.io/qt-5/implicit-sharing.html>.
8. Spanning Tree. (29/04/2024). Understanding B-Trees: The Data Structure Behind Modern Databases. YouTube. <https://www.youtube.com/watch?v=K1a2Bk8NrYQ>.
9. Hartmann, Andreas. (30/12/2024). Working with Btrfs – Snapshots. Fedora Magazine. <https://fedoramagazine.org/working-with-btrfs-snapshots/>.
10. BTRFS Wiki - Subvolumes (s.f.). <https://btrfs.readthedocs.io/en/latest/Subvolumes.html>
11. BTRFS Wiki - Scrub (s.f.). <https://btrfs.readthedocs.io/en/latest/Scrub.html>
12. ZFS vs BTRFS Comparison - Fedora Wiki (s.f.) <https://fedoraproject.org/wiki/Btrfs>

13. Comprendiendo los Sistemas de Archivos - Kingston (s.f.)
<https://www.kingston.com/latam/blog/personal-storage/understanding-file-systems>
14. Introducción a los árboles B - Geeksforgeeks (s.f.)
<https://www.geeksforgeeks.org/introduction-of-b-tree-2/>
15. GeeksForGeeks. (15/05/2020). Copy on Write.
<https://www.geeksforgeeks.org/copy-on-write/>.
16. Cfheoh. (01/10/2011). ONTAP vs ZFS.
<https://storagegaga.wordpress.com/2011/10/01/ontap-vs-zfs/>.
17. Shovon, Shahriar. (2021). How to Use Btrfs Snapshots.
<https://linuxhint.com/use-btrfs-snapshots/>.