18
Using DBFS

The DBFS File System implementation includes creating and accessing the file system and managing it.

- Enabling Advanced SecureFiles LOB Features for DBFS
  Using the `@dbfs_create_filesystem.sql` command, you can create a partitioned or non-partitioned file system with the compression and deduplicate options. If you want to specify additional options while creating the file system, use the `DBMS_DBFS_SFS.CREATEFILESYSTEM` procedure.
- Installing DBFS
  DBFS is a part of the Oracle Database installation.
- Creating a DBFS File System
  You can create a partitioned or non-partitioned DBFS File system.
- Accessing DBFS File System
  This section describes the various interfaces through which you can access the DBFS File System.
- Maintaining DBFS
  DBFS administration includes tools that perform diagnostics, manage failover, perform backup, and so on.
- Shrinking and Reorganizing DBFS Filesystems
  DBFS uses Online File system Reorganization to shrink itself, enabling the release of allocated space back to the containing tablespace.
- Dropping a File System
  You can drop a file system by running `DBFS_DROP_FILESYSTEM.SQL`.

18.1 Enabling Advanced SecureFiles LOB Features for DBFS

Using the `@dbfs_create_filesystem.sql` command, you can create a partitioned or non-partitioned file system with the compression and deduplicate options. If you want to specify additional options while creating the file system, use the `DBMS_DBFS_SFS.CREATEFILESYSTEM` procedure.

For information about all the additional options that you can use with the `DBMS_DBFS_SFS.CREATEFILESYSTEM` procedure, see CREATEFILESYSTEM Procedure in *PL/SQL Packages and Types Reference*.

Use the `@dbfs_create_filesystem.sql` command to quickly create, register, and mount a file system. When you use the `DBMS_DBFS_SFS.CREATEFILESYSTEM` procedure to enable additional options while creating a file system, you must additionally run commands to register and mount the file system that you create.

Let's use the `DBMS_DBFS_SFS.CREATEFILESYSTEM` procedure to create a file system with the encryption option.

Before you begin, ensure that you have created a wallet with the encryption key. See Administer Key Management in *SQL Language Reference*.

To create a file system with the encryption option:

1. Run the following command.
   Syntax

   ```
   exec
   dbms_dbfs_sfs.createFilesystem('store_name',tbl_tbs=>'tablespace_name',do_e
   ncrypt=> true | false,encryption=> encryption_type, do_dedup=> true |
   false,do_compress=>true | false);
   ```

   For reference information about the command options, see CREATEFILESYSTEM Procedure in *PL/SQL Packages and Types Reference*.

   Example

   For example, to create a file system in `Test3` store in the `test_fs1` tablespace with the default encryption, compression, and deduplicate options:

   ```
   exec dbms_dbfs_sfs.createFilesystem('test_fs1', tbl_tbs=>'Test3',
   do_encrypt=>true, encryption=>dbms_dbfs_sfs.ENCRYPTION_DEFAULT,
   do_dedup=>true, do_compress=>true);
   ```

   The file system is created with the option you have specified.

2. Run the following command to register the file system that you have created.

   Syntax

   ```
   dbms_dbfs_content.registerStore(store_name => 'filesystem_name',
   provider_name => 'posix',provider_package => 'dbms_dbfs_sfs') ;
   ```

   Example

   For example, run the following command to register the `test_fs1` file system.

   ```
   dbms_dbfs_content.registerStore(store_name => 'test_fs1', provider_name =>
   'posix', provider_package => 'dbms_dbfs_sfs') ;
   ```

3. Run the following command to mount the file system that you have created.

   Syntax

   ```
   dbms_dbfs_content.mountStore(store_name  => 'filesystem_name', store_mount
   => 'filesystem_name');
   ```

   Example

   For example, run the following command to mount the `test_fs1` file system.

   ```
   dbms_dbfs_content.mountStore(store_name  => 'test_fs1', store_mount =>
   'test_fs1');
   ```

18.7 Dropping a File System

You can drop a file system by running `DBFS_DROP_FILESYSTEM.SQL`.

> **Caution:**
>
> When you drop a file system, it deletes all the files and associated metadata. You won't be able to access the files.

1. Log in to the database instance:

   ```
   $ sqlplus dbfs_user/@db_server
   ```

2. Enter the following command:

   ```
   @$ORACLE_HOME/rdbms/admin/dbfs_drop_filesystem.sql file_system_name
   ```

When you drop a file system, it deletes all the files and associated metadata. You won't be able to access the files. If you want to access the file system after dropping a DBFS, you can restore the file system from a database backup or file system backup.

Depending on the backup policy in your organization, you may have a database backup or file system backup. To restore from a database backup, you'll have to restore the entire database and then use the restored file system. To restore the file system from a file system backup, create a new DBFS and restore the file system from the file system backup.

18.2 Installing DBFS

DBFS is a part of the Oracle Database installation.

`$ORACLE_HOME/rdbms/admin` contains these DBFS utility packages:

- Content API (CAPI)
- SecureFiles Store (SFS)

`$ORACLE_HOME/bin` contains:

- `dbfs_client` executable

`$ORACLE_HOME/rdbms/admin` contains:

- SQL (`.plb` extension) scripts for the content store

18.3 Creating a DBFS File System

You can create a partitioned or non-partitioned DBFS File system.

For both partitioned and non-partitioned DBFS, you can specify one or more of the following storage properties to specify how your files are stored in DBFS: compression and deduplication.

For example, you can configure DBFS as a compressed file system with partitioning. At the time of creating a DBFS file system, you must specify the set of features that you want to enable for the file system.

After creating a DBFS, you can track the usage of the DBFS. If you want to change the storage properties of the DBFS, you can reorganize the DBFS. You can update the metadata of the DBFS by changing the values for parameters, such as `deduplicate`, `compress`, and `partition`. For example, you may have created a DBFS to store all the files in the compressed format. If you want to change this property, you can reorganize the DBFS.

- Privileges Required to Create a DBFS File System
  Database users must certain privileges to create a file system.
- Creating a Non-Partitioned File System
  You can create a file system by running `DBFS_CREATE_FILESYSTEM.SQL` while logged in as a user with DBFS administrator privileges.
- Creating a Partitioned File System
  Files in DBFS are hash partitioned. Partitioning creates multiple physical segments in the database, and files are distributed randomly in these partitions.

18.3.1 Privileges Required to Create a DBFS File System

Database users must certain privileges to create a file system.

Following is the minimum set of privileges required for a database user to create a file system:

- `GRANT CONNECT`
- `CREATE SESSION`
- `RESOURCE, CREATE TABLE`
- `CREATE PROCEDURE`
- `DBFS_ROLE`

18.3.2 Creating a Non-Partitioned File System

You can create a file system by running `DBFS_CREATE_FILESYSTEM.SQL` while logged in as a user with DBFS administrator privileges.

Before you begin, ensure that you create the file system in an ASSM tablespace to support a SecureFile store.

To create a non-partitioned file system:

1. Log in to the database instance as a user with DBFS administrator privileges.

   ```
   $ sqlplus dbfs_user/@db_server
   ```

2. Enter the following command to create the file system.

   Syntax

   ```
   @$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem.sql tablespace_name
        file_system_name
     [compress-high | compress-medium | compress-low | nocompress]
     [deduplicate | nodeduplicate]
     non-partition
   ```

   Example

   For example, to create a file system called `staging_area` in an existing ASSM tablespace `dbfs_tbspc`:

```
$ sqlplus dbfs_user/db_server
   @$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem.sql
   dbfs_tbspc staging_area nocompress nodeduplicate non-partition
```

18.3.3 Creating a Partitioned File System

Files in DBFS are hash partitioned. Partitioning creates multiple physical segments in the database, and files are distributed randomly in these partitions.

You can create a partitioned file system by running `DBFS_CREATE_FILESYSTEM.SQL` while logged in as a user with DBFS administrator privileges.

The tablespace in which you create the file system should be an ASSM tablespace to support Securefile store. Before you begin, ensure that you create the file system in an ASSM tablespace to support SecureFile store.

1. Log in to the database instance:

   ```
   $ sqlplus dbfs_user/@db_server
   ```

2. Enter one of the following commands to create the file system based on your requirement.

   Syntax

   ```
   @$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem_advanced.sql tablespace_name
          file_system_name [compress-high | compress-medium | compress-low |
   nocompress]
     [deduplicate | nodeduplicate]
     [partition | partition-by-itemname | partition-by-guid | partition-by-path]
   ```

   Examples

   - For example, to create a partitioned file system called `staging_area` in an existing ASSM tablespace `dbfs_tbspc`:

     ```
     $ sqlplus dbfs_user/@db_server
            @$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem_advanced.sql dbfs_tbspc
            staging_area nocompress nodeduplicate partition
     ```

   - For example, to create a partitioned file system called `staging_area` in an existing ASSM tablespace `dbfs_tbspc` with the storage properties compress and deduplicate.

     ```
     $ sqlplus dbfs_user/@db_server
            @$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem_advanced.sql dbfs_tbspc
            staging_area compress-medium deduplicate partition
     ```

18.4 Accessing DBFS File System

This section describes the various interfaces through which you can access the DBFS File System.

- DBFS Client Prerequisites
  The DBFS File System client side application, which is named `dbfs_client`, runs on each system that will access to DBFS.
- Multiple Mount Points on DBFS Client
  Starting from Oracle Database Release 21c, a single Database File System (DBFS) client instance can mount multiple DBFS, owned by different database users across different database instances.

- Manager File System
  The Manager File System is the interface between the OS user and the DBFS Client. The OS user can communicate with the Client through limited File System commands.
- DBFS Client Command-Line Interface Operations
  The DBFS client command-line interface allows you to directly access files stored in DBFS.
- DBFS Mounting Interface (Linux and Solaris Only)
  You can mount DBFS using the `dbfs_client` in Linux and Solaris only.
- File System Security Model
  The database manages security in DBFS. It does not use the operating system security model.
- HTTP, WebDAV, and FTP Access to DBFS
  Components that enable HTTP, WebDAV, and FTP access to DBFS over the Internet use various XML DB server protocols.

18.4.1 DBFS Client Prerequisites

The DBFS File System client side application, which is named `dbfs_client`, runs on each system that will access to DBFS.

The prerequisites for the DBFS File System Client, `dbfs_client`, are:

- The `dbfs_client` host must have the Oracle client libraries installed.
- The `dbfs_client` can be used as a direct RDBMS client using the DBFS Command Interface on Linux, Linux.X64, Solaris, Solaris64, AIX, HPUX and Windows platforms.
- The `dbfs_client` can only be used as a mount client on Linux, Linux.X64, and Solaris 11 platforms. The `dbfs_client` host must have the `FUSE` Linux package or the Solaris `libfuse` package installed.

> **See Also:**
>
> DBFS Mounting Interface (Linux and Solaris Only) for further details.

The DBFS client command-line interface allows you to perform many pre-defined commands, such as copy files in and out of the DBFS filesystem from any host on the network.

The command-line interface has slightly better performance than the DBFS client mount interface because it does not mount the file system, thus bypassing the user space file system. However, it is not transparent to applications.

The DBFS client mount interface allows DBFS to be mounted through a file system mount point thus providing transparent access to files stored in DBFS with generic file system operations.

To run DBFS commands, specify `--command` to the DBFS client.

18.4.2 Multiple Mount Points on DBFS Client

Starting from Oracle Database Release 21c, a single Database File System (DBFS) client instance can mount multiple DBFS, owned by different database users across different database instances.

To enable access to multiple database users, the DBFS client has to manage multiple mount points. Each mount point enables one database user to access DBFS.

When the DBFS client provides access to a single database user through a single mount point, it is termed as Single User Mount Version (SUMV) mode and when the DBFS client provides access to multiple database users through multiple mount points, it is termed as Multi User Mount Version (MUMV) mode.

You can start a DBFS client in either of these modes. However, once you start the client in any mode, you cannot switch to the other mode without restarting the client. If a DBFS client is started in the MUMV mode, then the client creates a pseudo file system called Manager File System (MFS), which acts as an interface between the OS user and the DBFS client.

You can start the MUMV mode in two variants, one that can mount DBFS across multiple container databases or one that can mount only DBFS belonging to different pluggable databases of a single container database. The MUMV variant that mounts DBFS from multiple databases is termed as the Cross-Database variant and the one that mounts DBFS for multiple PDBs of a single container database as the CDB variant. Both the variants are started by specifying only the MFS mount points during start up. The DBFS mounts are added by setting extended attributes on the MFS mount point.

- MUMV for CDB Variant
  The CDB variant of the Multi User Mount Version (MUMV) mode manages the mount points of Database File System (DBFS) that belong to different pluggable databases (PDBs) of a single container database (CDB).
- MUMV for Cross-Database Variant
  The Cross-Database variant of the Multi User Mount Version (MUMV) mode manages mount points for Database File System (DBFS) in multiple databases.

18.4.2.1 MUMV for CDB Variant

The CDB variant of the Multi User Mount Version (MUMV) mode manages the mount points of Database File System (DBFS) that belong to different pluggable databases (PDBs) of a single container database (CDB).

Remember the following points while working with the CDB variant of the MUMV mode:

- The DBFS client, managing multiple DBFS mount points of a single container, should be provided with the credentials to connect to a common user of the CDB at `CDB$ROOT`. The DBFS to be mounted, should be created in or exported to this common user in the PDBs.
- A mount point must be specified for the DBFS in every PDB in the given container. The DBFS client connects to the `CDB$ROOT`, using common user credentials, and then switches to the required PDB to access the DBFS through the specified mount point.

18.4.2.2 MUMV for Cross-Database Variant

The Cross-Database variant of the Multi User Mount Version (MUMV) mode manages mount points for Database File System (DBFS) in multiple databases.

Remember the following points while working with the Cross-Database variant of the MUMV mode:

- The DBFS client must have the credentials of a database user on each database to manage the respective DBFS mount points.
- A DBFS mount point must be specified for each database user and a DBFS must be created in their respective schemas.

18.4.3 Manager File System

The Manager File System is the interface between the OS user and the DBFS Client. The OS user can communicate with the Client through limited File System commands.

The Manager File System (MFS) is enabled only in the Multi User Mount Version (MUMV) mode. It treats the various mount points managed by the DBFS Client as files. The MFS provides an easy interface for the OS users to manage multiple mount points.

The MFS does not create or store files on the disk. Only a limited file system operations are allowed on the MFS mount point.

No OS user can create files or directories under the MFS.

- Adding a DBFS Mount Point
  You can add DBFS mount points by specifying extended attributes on the MFS mount points.
- Listing DBFS Mount Points
  Each DBFS mount point has a corresponding file under the MFS directory, `/mnt/mfs`. So, you can use the standard Linux command `ls` to list the DBFS mount points.
- Unmounting a DBFS Mount Point
  The procedure to unmount a DBFS mount point is the same for both the CDB variant and the Cross-Database variant of the MUMV mode.
- Configuration Parameters of DBFS Client
  All configuration parameters of DBFS client in Single User Mount Version (SUMV) mode can also be used with the DBFS client in Multi User Mount Version (MUMV) mode at the time of start up.
- Diagnosability of DBFS Client
  Starting from Oracle Database Release 21c, the DBFS Client writes an alert file in the client trace directory of the configured Automatic Diagnostic Repository (ADR) base.

18.4.3.1 Adding a DBFS Mount Point

You can add DBFS mount points by specifying extended attributes on the MFS mount points.

> **Note:**
>
> The MUMV mode works only in wallet mode, even if you do not specify the `-o wallet` option. As there is no way to provide passwords in the DBFS commend-line interface, you must add all the credentials required by the DBFS client in the wallet.

While using a CDB variant of the MUMV mode, add the mount points for each of the PDB in the CDB by setting the extended attribute on the `/mnt/mfs` directory, where `/mnt/mfs` is the MFS mount point.

Defining the Mount Points in a CDB Variant

Perform the following steps to define the mount points in a CDB variant of the MUMV mode:

1. Start the DBFS client to connect to the common user at the `CDB$ROOT`, specifying the MFS mount point and the wallet alias at the start up:

   ```
   % dbfs_client -o mfs_mount=/mnt/mfs -o cdb=inst_cdb
   ```

   Where, `/mnt/mfs` is the MFS mount point. It can be any empty directory of your choice. `inst_cdb` is the alias insert into the wallet that can connect to the common user in `CDB$ROOT`.

2. Add a DBFS mount point by setting an extended attribute in the following way:

   ```
   % setfattr -n mount_pdb -v " pdb1 /mnt/mp1" /mnt/mfs/
   ```

   Where:

   - `mount_pdb` is the name of the extended attribute to mount a DBFS mount point in CDB variant
   - `pdb1` is the name of the PDB in the particular CDB, which is pointed to by `inst_cdb`
   - `/mnt/mp1` is the mount directory, where the DBFS present in the common user in the PDB `pdb1`, should be mounted
   - `/mnt/mfs` is the MFS mount directory that was used during the start up of the `dbfs_client` command

3. (Optional) Add more DBFS mount points by setting the same extended attribute with different arguments in the following way:

   ```
   % setfattr -n mount_pdb  -v " pdb2 /mnt/mp2" /mnt/mfs
   % setfattr -n mount_pdb  -v " pdb3 /mnt/mp3" /mnt/mfs
   ```

   Where, `pdb2` and `pdb3` are the actual names of the PDBs in the container.

Defining the Mount Points in a Cross-Database Variant

Perform the following steps to define the mount points in a Cross-Database variant of the MUMV mode:

1. Start the DBFS client in MUMV Cross-Database variant by specifying the MFS mount point at the start up in the following way:

   ```
   % dbfs_client -o mfs_mount=/mnt/mfs
   ```

   Where, `/mnt/mfs` is the MFS mount point. It can be any empty directory of your choice

2. Add a DBFS mount point by setting an extended attribute in the following way:

   ```
   % setfattr -n mount -v " inst1 /mnt/mp1" /mnt/mfs/
   ```

   Where,

   - `mount` is the name of the extended attribute to mount a DBFS mount in Cross-Database variant
   - `inst1` is the wallet alias that connects to the DB user, for which DBFS needs to be mounted

- `/mnt/mp1` is the mount directory, where the DBFS should be mounted
- `/mnt/mfs` is the MFS mount directory that was used during the start up of the `dbfs_client` command

3. (Optional) Add more DBFS mount points by setting the same extended attribute with different arguments in the following way:

```
% setfattr -n mount -v "inst2 /mnt/mp2" /mnt/mfs/
% setfattr -n mount -v "inst3 /mnt/mp3" /mnt/mfs/
```

Where, `inst2` and `inst3` are aliases that must exist in the wallet. The DBFS client must have the credentials to connect to the user in the database and they should have at least one DBFS created in their schema.

18.4.3.2 Listing DBFS Mount Points

Each DBFS mount point has a corresponding file under the MFS directory, `/mnt/mfs`. So, you can use the standard Linux command `ls` to list the DBFS mount points.

The following code snippet shows how to list all the DBFS mount points:

```
% ls  -l  /mnt/mfs
```

The content of each file under the `/mnt/mfs` directory, provides details about the parameters used in the corresponding mount point.

The MFS is a *read-only* file system. You cannot create any file or directory within it using any application, apart from the DBFS Client. Anything that appears as a file or a directory under the MFS, is defined by the DBFS Client.

18.4.3.3 Unmounting a DBFS Mount Point

The procedure to unmount a DBFS mount point is the same for both the CDB variant and the Cross-Database variant of the MUMV mode.

You must unmount a mount point using the FUSE executable file, `fusermount`. The following code snippet shows how to drop a DBFS mount point:

```
% fusermount -u /mnt/mp1
```

18.4.3.4 Configuration Parameters of DBFS Client

All configuration parameters of DBFS client in Single User Mount Version (SUMV) mode can also be used with the DBFS client in Multi User Mount Version (MUMV) mode at the time of start up.

All the command-line options passed to the DBFS client in the MUMV mode are inherited by all the DBFS mount points that may be added later. For example, for the following `dbfs_client` command, the DBFS mounted at the `/mnt/mp1` mount point automatically inherits the `spool_max` value as 32 and the `max_threads value` as 16:

```
% dbfs_client -o mfs_mount=/mnt/mfs -o spool_max=32 -o max_threads=16
% setfattr -n mount -v "inst1 /mnt/mp1" /mnt/mfs
```

If you want to configure a mount point differently than the DBFS client, then use the `setfattr` command in the following way:

```
% sefattr -n mount -v "inst2 /mnt/mp2 -o trace_file=/tmp/
clnt.trc,trace_level=1" /mnt/mfs
```

The preceding command enables only the trace for the DBFS client at the `/mnt/mp2` mount point, but does not inherit the `spool_max` and `max_threads` arguments that were specified at the time of start up. The values specified with the `setfattr` command overwrite the values specified during start up.

18.4.3.5 Diagnosability of DBFS Client

Starting from Oracle Database Release 21c, the DBFS Client writes an alert file in the client trace directory of the configured Automatic Diagnostic Repository (ADR) base.

The alert files are generated for every instance of the DBFS client and can be found under the `clients/DBFS/DBFS/trace` directory of the ADR base. The file name is of the format `dbfs_alert_<client_pid>.trc`.

The alert file is different from the trace file. It is always enabled and only important activities of the DBFS clients are written to the alert file.

> **Note:**
>
> The `diagnostic_dest` initialization parameter sets the location of the automatic diagnostic repository. When you use `dbfs_client` or Oracle File Server (OFS) as the file system server, ensure that this parameter does not point to a directory inside `dbfs_client` or OFS as this can produce a dependency cycle and cause the system to hang.

18.4.4 DBFS Client Command-Line Interface Operations

The DBFS client command-line interface allows you to directly access files stored in DBFS.

- About the DBFS Client Command-Line Interface
  The DBFS client command-line interface allows you to perform many pre-defined commands, such as copy files in and out of the DBFS filesystem from any host on the network.
- Listing a Directory
  You can use the `ls` command to list the contents of a directory.
- Copying Files and Directories
  You can use the `cp` command to copy files or directories from the source location to the destination location.
- Removing Files and Directories
  You can use the command `rm` to delete a file or directory.

18.4.4.1 About the DBFS Client Command-Line Interface

The DBFS client command-line interface allows you to perform many pre-defined commands, such as copy files in and out of the DBFS filesystem from any host on the network.

The command-line interface has slightly better performance than the DBFS client mount interface because it does not mount the file system, thus bypassing the user space file system. However, it is not transparent to applications.

The DBFS client mount interface allows DBFS to be mounted through a file system mount point thus providing transparent access to files stored in DBFS with generic file system operations.

To run DBFS commands, specify `--command` to the DBFS client.

All DBFS content store paths , in command-line interface ,must be preceded by `dbfs:` .This is an example: `dbfs:/staging_area/file1`. All database path names specified must be absolute paths.

```
dbfs_client db_user@db_server--command command [switches] [arguments]
```

where:

- `command` is the executable command, such as `ls, cp, mkdir,` or `rm`.
- `switches` are specific for each command.
- `arguments` are file names or directory names, and are specific for each command.

Note that dbfs_client returns a nonzero value in case of failure.

18.4.4.2 Listing a Directory

You can use the `ls` command to list the contents of a directory.

Use this syntax:

```
dbfs_client db_user@db_server --command ls [switches] target
```

where

- *`target`* is the listed directory.
- *`switches`* is any combination of the following:
  - *`-a`* shows all files, including '.' and '..'.
  - *`-l`* shows the long listing format: name of each file, the file type, permissions, and size.
  - *`-R`* lists subdirectories recursively.

For example:

```
$ dbfs_client ETLUser@DBConnectString --command  ls dbfs:/staging_area/dir1
```

or

```
$ dbfs_client ETLUser@DBConnectString --command  ls -l -a -R dbfs:/staging_area/dir1
```

18.4.4.3 Copying Files and Directories

You can use the `cp` command to copy files or directories from the source location to the destination location.

The `cp` command also supports recursive copy of directories.

```
dbfs_client db_user@db_server --command cp [switches] source destination
```

where:

- *source* is the source location.
- *destination* is the destination location.
- *switches* is either -R or -r, the options to recursively copy all source contents into the destination directory.

The following example copies the contents of the local directory, 01-01-10-dump recursively into a directory in DBFS:

```
$ dbfs_client ETLUser@DBConnectString --command cp -R  01-01-10-dump dbfs:/staging_area/
```

The following example copies the file hello.txt from DBFS to a local file Hi.txt:

```
$ dbfs_client ETLUser@DBConnectString --command cp dbfs:/staging_area/hello.txt Hi.txt
```

18.4.4.4 Removing Files and Directories

You can use the command rm to delete a file or directory.

The command rm also supports recursive delete of directories.

```
dbfs_client db_user@db_server --command rm [switches] target
```

where:

- *target* is the listed directory.
- *switches* is either -R or -r, the options to recursively delete all contents.

For example:

```
$ dbfs_client ETLUser@DBConnectString --command rm  dbfs:/staging_area/srcdir/hello.txt
```

or

```
$ dbfs_client ETLUser@DBConnectString --command rm -R  dbfs:/staging_area/dir1
```

18.4.5 DBFS Mounting Interface (Linux and Solaris Only)

You can mount DBFS using the dbfs_client in Linux and Solaris only.

The instructions indicate the different requirements for the Linux and Solaris platforms.

- Installing FUSE on Solaris 11 SRU7 and Later
  You can use dbfs_client as a mount client in Solaris 11 SRU7 and later, if you install FUSE
- Solaris-Specific Privileges
  On Solaris, the user must have the Solaris privilege PRIV_SYS_MOUNT to perform mount and unmount operations on DBFS filesystems.
- About the Mount Command for Solaris and Linux
  The dbfs_client mount command for Solaris and Linux uses specific syntax.
- Mounting a File System with a Wallet
  You can mount a file system with a wallet after configuring various environment variables.
- Mounting a File System with Password at Command Prompt
  You must enter a password at the command prompt to mount a file system using dbfs_client.

- Unmounting a File System
  In Linux, you can run `fusermount` to unmount file systems.
- Mounting DBFS Through fstab Utility for Linux
  In Linux, you can configure `fstab` utility to use `dbfs_client` to mount a DBFS filesystem.
- Mounting DBFS Through the vfstab Utility for Solaris
  On Solaris, file systems are commonly configured using the `vfstab` utility.
- Restrictions on Mounted File Systems
  DBFS supports most file system operations with exceptions.
- Restrictions on Types of Files Stored at DBFS Mount Points
  DBFS should be avoided in scenarios that can cause a file operation on the DBFS files resulting in more data to be written back to the DBFS.

18.4.5.1 Installing FUSE on Solaris 11 SRU7 and Later

You can use `dbfs_client` as a mount client in Solaris 11 SRU7 and later, if you install `FUSE`

Install `FUSE` to use `dbfs_client` as a mount client in Solaris 11 SRU7 and later.

- Run the following package as `root`.

```
pkg install libfuse
```

18.4.5.2 Solaris-Specific Privileges

On Solaris, the user must have the Solaris privilege `PRIV_SYS_MOUNT` to perform mount and unmount operations on DBFS filesystems.

Give the user the Solaris privilege `PRIV_SYS_MOUNT` .

1. Edit `/etc/user_attr`.
2. Add or modify the user entry (assuming the user is Oracle) as follows:

```
oracle::::type=normal;project=group.dba;defaultpriv=basic,priv_sys_mount;;auth
s=solaris.smf.*
```

18.4.5.3 About the Mount Command for Solaris and Linux

The `dbfs_client` mount command for Solaris and Linux uses specific syntax.

Syntax:

```
dbfs_client db_user@db_server [-o option_1 -o option_2 ...] mount_point
```

where the mandatory parameters are:

- *`db_user`* is the name of the database user who owns the DBFS content store file system.
- *`db_server`* is a valid connect string to the Oracle Database server, such as `hrdb_host:1521/hrservice` or an alias specified in the `tnsnames.ora`.
- *`mount_point`* is the path where the Database File System is mounted. Note that all file systems owned by the database user are visible at the mount point.

The options are:

- `direct_io`: To bypass the OS page cache and provide improved performance for large files. Programs in the file system cannot be executed with this option. Oracle recommends this option when DBFS is used as an ETL staging area.
- `wallet`: To run the DBFS client in the background. The Wallet must be configured to get its credentials.
- `failover`: To fail over the DBFS client to surviving database instances without data loss. Expect some performance cost on writes, especially for small files.
- `allow_root`: To allow the root user to access the filesystem. You must set the `user_allow_other` parameter in the `/etc/fuse.conf` configuration file.
- `allow_other`: To allow other users to access the filesystem. You must set the `user_allow_other` parameter in the `/etc/fuse.conf` configuration file.
- `rw`: To mount the filesystem as read-write. This is the default setting.
- `ro`: To mount the filesystem as read-only. Files cannot be modified.
- `trace_level=`*`n`* sets the trace level. Trace levels are:
  - `1 DEBUG`
  - `2 INFO`
  - `3 WARNING`
  - `4 ERROR`: The default tracing level. It outputs diagnostic information only when an error happens. It is recommended that this tracing level is always enabled.
  - `5 CRITICAL`
- `trace_file=STR`: Specifies the tracing log file, where `STR` can be either a *`file_name`* or `syslog`.
- `trace_size=`*`trcfile_size`*: Specifies size of the trace file in MB. By default, `dbfs_client` rotates tracing output between two 10MB files. Specifying `0` for `trace_size` sets the maximum size of the trace file to unlimited.

18.4.5.4 Mounting a File System with a Wallet

You can mount a file system with a wallet after configuring various environment variables.

You must first configure the `LD_LIBRARY_PATH`, `ORACLE_HOME` environment variables and `sqlnet.ora` correctly before mounting a file system with a wallet.

1. Login as admin user.
2. Mount the DBFS store. (Oracle recommends that you do not perform this step as root user.)

   ```
   % dbfs_client @/dbfsdb -o wallet,rw,user,direct_io /mnt/dbfs
   ```

3. [Optional] To test if the previous step was successful, as admin user, list the `dbfs` directory.

   ```
   $ ls /mnt/tdbfs
   ```

   Using the wallet option runs the `dbfs_client` in the background

> **See Also:**
>
> Using Oracle Wallet with DBFS Client

18.4.5.5 Mounting a File System with Password at Command Prompt

You must enter a password at the command prompt to mount a file system using `dbfs_client`.

- Run the following command at the command prompt and provide the password:

```
$ dbfs_client ETLUser@DBConnectString /mnt/dbfs
  password: xxxxxxx
```

The `dbfs_client` runs in the foreground after the password is provided at the command prompt.

18.4.5.6 Unmounting a File System

In Linux, you can run `fusermount` to unmount file systems.

- Linux
- Solaris

Linux

To run `fusermount` in Linux, do the following:

- Run the following:

```
$ fusermount -u <mount point>
```

Solaris

In Solaris, you can run `umount` to unmount file systems.

- Run the following:

```
$ umount -u <mount point>
```

18.4.5.7 Mounting DBFS Through fstab Utility for Linux

In Linux, you can configure `fstab` utility to use `dbfs_client` to mount a DBFS filesystem.

To mount DBFS through `/etc/fstab`, you must use Oracle Wallet for authentication.

1. Login as `root` user.
2. Change the user and group of `dbfs_client` to user `root` and group `fuse`.

```
# chown root.fuse $ORACLE_HOME/bin/dbfs_client
```

3. Set the `setuid` bit on `dbfs_client` and restrict `execute` privileges to the user and group only.

   ```
   # chmod u+rwxs,g+rx-w,o-rwx dbfs_client
   ```

4. Create a symbolic link to `dbfs_client` in `/sbin` as "`mount.dbfs`".

   ```
   $ ln -s $ORACLE_HOME/bin/dbfs_client /sbin/mount.dbfs
   ```

5. Create a new Linux group called "`fuse`".
6. Add the Linux user that is running the DBFS Client to the `fuse` group.
7. Add the following line to `/etc/fstab`:

   ```
   /sbin/mount.dbfs#db_user@db_server mount_point fuse rw,user,noauto 0 0
   ```

   For example:

   ```
   /sbin/mount.dbfs#/@DBConnectString /mnt/dbfs fuse rw,user,noauto 0 0
   ```

8. The Linux user can mount the DBFS file system using the standard Linux `mount` command. For example:

   ```
   $ mount /mnt/dbfs
   ```

   Note that `FUSE` does not currently support `automount`.

18.4.5.8 Mounting DBFS Through the vfstab Utility for Solaris

On Solaris, file systems are commonly configured using the `vfstab` utility.

1. Create a mount shell script `mount_dbfs.sh` to use to start `dbfs_client`. All the environment variables that are required for Oracle RDBMS must be exported. These environment variables include `TNS_ADMIN`, `ORACLE_HOME`, and `LD_LIBRARY_PATH`. For example:

   ```
   #!/bin/ksh
   export TNS_ADMIN=/export/home/oracle/dbfs/tnsadmin
   export ORACLE_HOME=/export/home/oracle/11.2.0/dbhome_1
   export DBFS_USER=dbfs_user
   export DBFS_PASSWD=/tmp/passwd.f
   export DBFS_DB_CONN=dbfs_db
   export O=$ORACLE_HOME
   export LD_LIBRARY_PATH=$O/lib:$O/rdbms/lib:/usr/lib:/lib:$LD_LIBRARY_PATH
   export NOHUP_LOG=/tmp/dbfs.nohup

   (nohup $ORACLE_HOME/bin/dbfs_client $DBFS_USER@$DBFS_DB_CONN < $DBFS_PASSWD
            2>&1 & ) &
   ```

2. Add an entry for DBFS to `/etc/vfstab`. Specify the `mount_dbfs.sh` script for the `device_to_mount`. Specify `uvfs` for the `FS_type`. Specify no `formount_at_boot`. Specify mount options as needed. For example:

   ```
   /usr/local/bin/mount_dbfs.sh - /mnt/dbfs uvfs - no rw,allow_other
   ```

3. User can mount the DBFS file system using the standard Solaris mount command. For example:

   ```
   $ mount /mnt/dbfs
   ```

4. User can unmount the DBFS file system using the standard Solaris umount command. For example:

   ```
   $ umount /mnt/dbfs
   ```

18.4.5.9 Restrictions on Mounted File Systems

DBFS supports most file system operations with exceptions.

The exceptions are:

- `ioctl`
- range locking (file locking is supported)
- asynchronous I/O through `libaio`
- `O_DIRECT` file opens
- hard links
- other special file modes

Memory-mapped files are supported except in shared-writable mode. For performance reasons, DBFS does not update the file access time every time file data or the file data attributes are read.

> **Note:**
>
> You should not run programs from a DBFS-mounted file system which was mounted with the `direct_io` option.

Oracle does not support exporting DBFS file systems using NFS or Samba.

18.4.5.10 Restrictions on Types of Files Stored at DBFS Mount Points

DBFS should be avoided in scenarios that can cause a file operation on the DBFS files resulting in more data to be written back to the DBFS.

The following scenarios are not exhaustive but provide examples of operations that can make the DBFS and the database interdependent and hence should be avoided:

- **Sample Scenario 1:** DBFS is the destination for the trace files generated by the same database that is hosting the DBFS.
  For example: The act of writing the trace file into the DBFS could generate more trace data to be written back into DBFS.
- **Sample Scenario 2:** The trail file of a database replication is in a DBFS and the DBFS is in the SAME database that is being replicated.
  For example: The act of writing into the trail by the replication process generates redo. This redo could feed back into the replication.
- **Sample Scenario 3:** DBFS is the destination of any database files of the same database.
  For example: The data files, control files, redo log files could make the DBFS and the database inter dependent.

18.4.6 File System Security Model

The database manages security in DBFS. It does not use the operating system security model.

- About the File System Security Model
  DBFS operates under a security model where all file systems created by a user are private to that user, by default.

- Enabling Shared Root Access
  As an operating system user who mounts the file system, you can allow root access to the file system by specifying the `allow_root` option.
- About DBFS Access Among Multiple Database Users
  DBFS allows multiple users to share a subset of the filesystem state.
- Establishing DBFS Access Sharing Across Multiple Database Users
  Learn about sharing access of DBFS to multiple database users in this section.

18.4.6.1 About the File System Security Model

DBFS operates under a security model where all file systems created by a user are private to that user, by default.

Oracle recommends maintaining this model. Because operating system users and Oracle Database users are different, it is possible to allow multiple operating system users to mount a single DBFS filesystem. These mounts may potentially have different mount options and permissions. For example, OS `user1` may mount a DBFS filesystem as `READ ONLY`, and OS `user2` may mount it as `READ WRITE`. However, Oracle Database views both users as having the same privileges because they would be accessing the filesystem as the same database user.

Access to a database file system requires a database login as a database user with privileges on the tables that underlie the file system.The database administrator grants access to a file system to database users, and different database users may have different `READ` or `UPDATE` privileges to the file system. The database administrator has access to all files stored in the DBFS file system.

On each client computer, access to a DBFS mount point is limited to the operating system user that mounts the file system. This, however, does not limit the number of users who can access the DBFS file system, because many users may separately mount the same DBFS file system.

DBFS only performs database privilege checking. Linux performs operating system file-level permission checking when a DBFS file system is mounted. DBFS does not perform this check either when using the command interface or when using the PL/SQL interface directly.

18.4.6.2 Enabling Shared Root Access

As an operating system user who mounts the file system, you can allow root access to the file system by specifying the `allow_root` option.

This option requires that the `/etc/fuse.conf` file contain the `user_allow_other` field, as demonstrated in Example 18-1.

Example 18-1 Enabling Root Access for Other Users

```
# Allow users to specify the 'allow_root' mount option.
user_allow_other
```

18.4.6.3 About DBFS Access Among Multiple Database Users

DBFS allows multiple users to share a subset of the filesystem state.

A Single filesystem may be accessed by multiple database users. For example, the database user that owns the filesystem may be a privileged user and sharing its user credentials may pose a security risk. To mitigate this, DBFS allows multiple database users to share a subset of the filesystem state.

While DBFS registrations and mounts made through the DBFS Content API are private to each user, the underlying filesystem and the tables on which they rely may be shared across users. After this is done, the individual filesystems may be independently mounted and used by different database users, either through SQL/PLSQL, or through `dbfs_client`.

18.4.6.4 Establishing DBFS Access Sharing Across Multiple Database Users

Learn about sharing access of DBFS to multiple database users in this section.

In the following example, user `user1` is able to modify the filesystem, and user `user2` can see these changes. Here, `user1` is the database user that creates a filesystem, and `user2` is the database user that eventually uses `dbfs_client` to mount and access the filesystem. Both `user1` and `user2` must have the `DBFS_ROLE` privilege.

1. Connect as the user who creates the filesystem.

```
sys@tank as sysdba> connect user1
Connected.
```

2. Create the filesystem `user1_FS`, register the store, and mount it as `user1_mt`.

```
user1@tank> exec dbms_dbfs_sfs.createFilesystem('user1_FS');
user1@tank> exec dbms_dbfs_content.registerStore('user1_FS', 'posix',
'DBMS_DBFS_SFS');
user1@tank> exec dbms_dbfs_content.mountStore('user1_FS', 'user1_mnt');
user1@tank> commit;
```

3. [Optional] You may check that the previous step has completed successfully by viewing all mounts.

```
user1@tank> select * from table(dbms_dbfs_content.listMounts);

STORE_NAME           |   STORE_ID|PROVIDER_NAME
---------------------|- ---------|----------------------------------------
PROVIDER_PKG         |PROVIDER_ID|PROVIDER_VERSION     |STORE_FEATURES
---------------------|-----------|---------------------|--------------------
STORE_GUID
----------
STORE_MOUNT
--------------------------------------------------------------------------------
CREATED
--------------------------------------------------------------------------------
MOUNT_PROPERTIES(PROPNAME, PROPVALUE, TYPECODE)
--------------------------------------------------------------------------------
user1_FS             | 1362968596|posix
"DBMS_DBFS_SFS"      | 3350646887|0.5.0               | 12714135  141867344
user1_mnt
01-FEB-10 09.44.25.357858 PM
DBMS_DBFS_CONTENT_PROPERTIES_T(
  DBMS_DBFS_CONTENT_PROPERTY_T('principal', (null), 9),
  DBMS_DBFS_CONTENT_PROPERTY_T('owner', (null), 9),
  DBMS_DBFS_CONTENT_PROPERTY_T('acl', (null), 9),
  DBMS_DBFS_CONTENT_PROPERTY_T('asof', (null), 187),
  DBMS_DBFS_CONTENT_PROPERTY_T('read_only', '0', 2))
```

4. [Optional] Connect as the user who will use the `dbfs_client`.

```
user1@tank> connect user2
Connected.
```

5. [Optional] Note that `user2` cannot see `user1`'s DBFS state, as `user2` has no mounts.

```
user2@tank> select * from table(dbms_dbfs_content.listMounts);
```

6. While connected as `user1`, export filesystem `user1_FS` for access to any user with `DBFS_ROLE` privilege.

```
user1@tank> exec dbms_dbfs_sfs.exportFilesystem('user1_FS');
user1@tank> commit;
```

7. Connect as the user who will use the `dbfs_client`.

```
user1@tank> connect user2
Connected.
```

8. As `user2`, view all available tables.

```
user2@tank> select * from table(dbms_dbfs_sfs.listTables);

SCHEMA_NAME                |TABLE_NAME                 |PTABLE_NAME
---------------------------|---------------------------|-------------------
VERSION#
--------------------------------CREATED
--------------------------------------------------------------------------------
FORMATTED
--------------------------------------------------------------------------------
PROPERTIES(PROPNAME, PROPVALUE, TYPECODE)
--------------------------------------------------------------------------------
user1                        |SFS$_FST_11               |SFS$_FSTP_11
0.5.0
01-FEB-10 09.43.53.497856 PM
01-FEB-10 09.43.53.497856 PM
(null)
```

9. As `user2`, register and mount the store, but do not re-create the `user1_FS` filesystem.

```
user2@tank> exec dbms_dbfs_sfs.registerFilesystem(
   'user2_FS', 'user1', 'SFS$_FST_11');
user2@tank> exec dbms_dbfs_content.registerStore(
   'user2_FS', 'posix', 'DBMS_DBFS_SFS');
user2@tank> exec dbms_dbfs_content.mountStore(
   'user2_FS', 'user2_mnt');
user2@tank> commit;
```

10. [Optional] As `user2`, you may check that the previous step has completed successfully by viewing all mounts.

```
user2@tank> select * from table(dbms_dbfs_content.listMounts);

STORE_NAME           |   STORE_ID|PROVIDER_NAME
---------------------|- ---------|----------------------------------------
PROVIDER_PKG         |PROVIDER_ID|PROVIDER_VERSION     |STORE_FEATURES
---------------------|-----------|---------------------|--------------------
STORE_GUID
----------
STORE_MOUNT
--------------------------------------------------------------------------------
CREATED
--------------------------------------------------------------------------------
MOUNT_PROPERTIES(PROPNAME, PROPVALUE, TYPECODE)
--------------------------------------------------------------------------------
user2_FS             | 1362968596|posix
"DBMS_DBFS_SFS"      | 3350646887|0.5.0               | 12714135  141867344
user1_mnt
01-FEB-10 09.46.16.013046 PM
DBMS_DBFS_CONTENT_PROPERTIES_T(
  DBMS_DBFS_CONTENT_PROPERTY_T('principal', (null), 9),
```

```
  DBMS_DBFS_CONTENT_PROPERTY_T('owner', (null), 9),
  DBMS_DBFS_CONTENT_PROPERTY_T('acl', (null), 9),
  DBMS_DBFS_CONTENT_PROPERTY_T('asof', (null), 187),
  DBMS_DBFS_CONTENT_PROPERTY_T('read_only', '0', 2))
```

11. [Optional] List path names for `user2` and `user1`. Note that another mount, `user2_mnt`, for store `user2_FS`, is available for `user2`. However, the underlying filesystem data is the same for `user2` as for `user1`.

```
user2@tank> select pathname from dbfs_content;

PATHNAME
--------------------------------------------------------------------------------
/user2_mnt
/user2_mnt/.sfs/tools
/user2_mnt/.sfs/snapshots
/user2_mnt/.sfs/content
/user2_mnt/.sfs/attributes
/user2_mnt/.sfs/RECYCLE
/user2_mnt/.sfs

user2@tank> connect user1
Connected.

user1@tank> select pathname from dbfs_content;

PATHNAME
--------------------
/user1_mnt
/user1_mnt/.sfs/tools
/user1_mnt/.sfs/snapshots
/user1_mnt/.sfs/content
/user1_mnt/.sfs/attributes
/user1_mnt/.sfs/RECYCLE
/user1_mnt/.sfs
```

12. In filesystem `user1_FS`, `user1` creates file *xxx*.

```
user1@tank> declare
                data blob;
                properties dbms_dbfs_content.properties_t;
            begin
                properties('posix:mode') :=
dbms_dbfs_content.propNumber(33188);
                dbms_dbfs_content.createFile('/user1_mnt/xxx', properties
=> properties, content => data);
            end;
            /
```

13. [Optional] Write to file *xxx*, created in the previous step.

```
user1@tank> var buf varchar2(100);
user1@tank> exec :buf := 'hello world';
user1@tank> exec dbms_lob.writeappend(:data, length(:buf),
utl_raw.cast_to_raw(:buf));
user1@tank> commit;
```

14. [Optional] Show that file *xxx* exists, and contains the appended data.

```
user1@tank> select pathname, utl_raw.cast_to_varchar2(filedata)
  from dbfs_content where filedata is not null;

PATHNAME
--------------------------------------------------------------------------------
UTL_RAW.CAST_TO_VARCHAR2(FILEDATA)
--------------------------------------------------------------------------------
/user1_mnt/xxx
hello world
```

15. User `user2` sees the same file in their own DBFS-specific path name and mount prefix.

```
user1@tank> connect user2
Connected.

user2@tank> select pathname, utl_raw.cast_to_varchar2(filedata) from
  dbfs_content where filedata is not null;

PATHNAME
--------------------------------------------------------------------------------
UTL_RAW.CAST_TO_VARCHAR2(FILEDATA)
--------------------------------------------------------------------------------
/user2_mnt/xxx
hello world
```

After the export and register pairing completes, both users behave as equals with regard to their usage of the underlying tables. The `exportFilesystem()` procedure manages the necessary grants for access to the same data, which is shared between schemas. After `user1` calls `exportFilesystem()`, filesystem access may be granted to any user with `DBFS_ROLE`. Note that a different role can be specified to `exportFilesystem`.

Subsequently, `user2` may create a new DBFS filesystem that shares the same underlying storage as the `user1_FS` filesystem, by invoking `dbms_dbfs_sfs.registerFilesystem()`, `dbms_dbfs_sfs.registerStore()`, and `dmbs_dbfs_sfs.mountStore()` procedure calls.

When multiple database users share a filesystem, they must ensure that all database users unregister their interest in the filesystem before the owner (here, `user1`) drops the filesystem.

Oracle does not recommend that you run the DBFS as `root`.

18.4.7 HTTP, WebDAV, and FTP Access to DBFS

Components that enable HTTP, WebDAV, and FTP access to DBFS over the Internet use various XML DB server protocols.

- Internet Access to DBFS Through XDB
  To provide database users who have DBFS authentication with a hierarchical file system-like view of registered and mounted DBFS stores, stores are displayed under the path `/dbfs`.
- Web Distributed Authoring and Versioning (WebDAV) Access
  WebDAV is an IETF standard protocol that provides users with a file-system-like interface to a repository over the Internet.
- FTP Access to DBFS
  FTP access to DBFS uses the standard FTP clients found on most Unix-based distributions. FTP is a file transfer mechanism built on client-server architecture with separate control and data connections.
- HTTP Access to DBFS
  Users have read-only access through HTTP/HTTPS protocols.

18.4.7.1 Internet Access to DBFS Through XDB

To provide database users who have DBFS authentication with a hierarchical file system-like view of registered and mounted DBFS stores, stores are displayed under the path `/dbfs`.

The `/dbfs` folder is a virtual folder because the resources in its subtree are stored in DBFS stores, not the XDB repository. XDB issues a `dbms_dbfs_content.list()` command for the root path name "`/`" (with invoker rights) and receives a list of store access points as subfolders in the `/dbfs` folder. The list is comparable to `store_mount` parameters passed to `dbms_dbfs_content.mountStore()`. FTP and WebDAV users can navigate to these stores, while HTTP and HTTPS users access URLs from browsers.

Note that features implemented by the XDB repository, such as repository events, resource configurations, and ACLs, are not available for the `/dbfs` folder.

DBFS Content API for guidelines on DBFS store creation, registration, deregistration, mount, unmount and deletion

18.4.7.2 Web Distributed Authoring and Versioning (WebDAV) Access

WebDAV is an IETF standard protocol that provides users with a file-system-like interface to a repository over the Internet.

WebDAV server folders are typically accessed through Web Folders on Microsoft Windows (2000/NT/XP/Vista/7, and so on). You can access a resource using its fully qualified name, for example, `/dbfs/sfs1/dir1/file1.txt`, where `sfs1` is the name of a DBFS store.

You need to set up WebDAV on Windows to access the DBFS filesystem.

> **See Also:**
>
> *Oracle XML DB Developer's Guide*

The user authentication required to access the DBFS virtual folder is the same as for the XDB repository.

When a WebDAV client connects to a WebDAV server for the first time, the user is typically prompted for a username and password, which the client uses for all subsequent requests. From a protocol point-of-view, every request contains authentication information, which XDB uses to authenticate the user as a valid database user. If the user does not exist, the client does not get access to the DBFS store or the XDB repository. Upon successful authentication, the database user becomes the current user in the session.

XDB supports both basic authentication and digest authentication. For security reasons, it is highly recommended that HTTPS transport be used if basic authentication is enabled.

18.4.7.3 FTP Access to DBFS

FTP access to DBFS uses the standard FTP clients found on most Unix-based distributions. FTP is a file transfer mechanism built on client-server architecture with separate control and data connections.

FTP users are authenticated as database users. The protocol, as outlined in RFC 959, uses clear text user name and password for authentication. Therefore, FTP is not a secure protocol.

The following commands are supported for DBFS:

- `USER`: Authentication username
- `PASS`: Authentication password
- `CWD`: Change working directory
- `CDUP`: Change to Parent directory
- `QUIT`: Disconnect
- `PORT`: Specifies an address and port to which the server should connect
- `PASV`: Enter passive mode
- `TYPE`: Sets the transfer mode, such as, ASCII or Binary
- `RETR`: Transfer a copy of the file
- `STOR`: Accept the data and store the data as a file at the server site
- `RNFR`: Rename From
- `RNTO`: Rename To
- `DELE`: Delete file
- `RMD`: Remove directory
- `MKD`: Make a directory
- `PWD`: Print working directory
- `LIST`: Listing of a file or directory. Default is current directory.
- `NLST`: Returns file names in a directory
- `HELP`: Usage document
- `SYST`: Return system type
- `FEAT`: Gets the feature list implemented by the server
- `NOOP`: No operation (used for keep-alives)
- `EPRT`: Extended address (IPv6) and port to which the server should connect
- `EPSV`: Enter extended passive mode (IPv6)

18.4.7.4 HTTP Access to DBFS

Users have read-only access through HTTP/HTTPS protocols.

Users point their browsers to a DBFS store using the XDB HTTP server with a URL such as `https://hostname:port/dbfs/sfs1` where `sfs1` is a DBFS store name.

18.5 Maintaining DBFS

DBFS administration includes tools that perform diagnostics, manage failover, perform backup, and so on.

- Using Oracle Wallet with DBFS Client
  Learn about using Oracle Wallet in this section.

- DBFS Diagnostics
  The `dbfs_client` program supports multiple levels of tracing to help diagnose problems.
- Preventing Data Loss During Failover Events
  The `dbfs_client` program can failover to one of the other existing database instances if one of the database instances in an Oracle RAC cluster fails.
- Bypassing Client-Side Write Caching
  The sharing and caching semantics for `dbfs_client` are similar to NFS in using the *close-to-open cache consistency* behavior.
- Backing up DBFS
  You have two alternatives for backing up DBFS.
- Small File Performance of DBFS
  Like any shared file system, the performance of DBFS for small files lags the performance of a local file system.

18.5.1 Using Oracle Wallet with DBFS Client

Learn about using Oracle Wallet in this section.

An Oracle Wallet allows the DBFS client to mount a DBFS store without requiring the user to enter a password.

> **See Also:**
>
> *Oracle Database Enterprise User Security Administrator's Guide* for more information about creation and management of wallets. Enterprise User Security (EUS) is deprecated with Oracle Database 23ai.

1. Create a directory for the wallet. For example:

   ```
   mkdir $ORACLE_HOME/oracle/wallet
   ```

2. Create an auto-login wallet.

   ```
   mkstore -wrl $ORACLE_HOME/oracle/wallet -create
   ```

   The `mkstore` wallet management command line tool is deprecated with Oracle Database 23ai, and can be removed in a future release.

3. Add the wallet location in the client's `sqlnet.ora` file:

   ```
   WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
    $ORACLE_HOME/oracle/wallet) ) )
   ```

4. Add the following parameter in the client's `sqlnet.ora` file:

   ```
   SQLNET.WALLET_OVERRIDE = TRUE
   ```

5. Create credentials:

   ```
   mkstore -wrl wallet_location -createCredential db_connect_string username password
   ```

   For example:

   ```
   mkstore -wrl $ORACLE_HOME/oracle/wallet -createCredential DBConnectString scott
   password
   ```

6. Add the connection alias to your `tnsnames.ora` file.

7. Use `dbfs_client` with Oracle Wallet.

   For example:

   ```
   $ dbfs_client -o wallet /@DBConnectString /mnt/dbfs
   ```

18.5.2 DBFS Diagnostics

The `dbfs_client` program supports multiple levels of tracing to help diagnose problems.

The `dbfs_client` can either output traces to a file or to `/var/log/messages` using the `syslog` daemon on Linux.

> **Note:**
>
> The `diagnostic_dest` initialization parameter sets the location of the automatic diagnostic repository. When you use `dbfs_client` or Oracle File Server (OFS) as the file system server, ensure that this parameter does not point to a directory inside `dbfs_client` or OFS as this can produce a dependency cycle and cause the system to hang.

When you trace to a file, the `dbfs_client` program keeps two trace files on disk. `dbfs_client`, rotates the trace files automatically, and limits disk usage to 10 MB.

By default, tracing is turned off except for critical messages which are always logged to `/var/log/messages`.

If `dbfs_client` cannot connect to the Oracle Database, enable tracing using the `trace_level` and `trace_file` options. Tracing prints additional messages to log file for easier debugging.

DBFS uses Oracle Database for storing files. Sometimes Oracle server issues are propagated to `dbfs_client` as errors. If there is a `dbfs_client` error, please view the Oracle server logs to see if that is the root cause.

18.5.3 Preventing Data Loss During Failover Events

The `dbfs_client` program can failover to one of the other existing database instances if one of the database instances in an Oracle RAC cluster fails.

For `dbfs_client` failover to work correctly, you must modify the Oracle database service and specify failover parameters. Run the `DBMS_SERVICE.MODIFY_SERVICE` procedure to modify the service as shown Example 18-2

Example 18-2 Enabling DBFS Client Failover Events

```
exec DBMS_SERVICE.MODIFY_SERVICE(service_name => 'service_name',
              aq_ha_notifications => true,
              failover_method => 'BASIC',
              failover_type => 'SELECT',
              failover_retries => 180,
              failover_delay => 1);
```

Once you have completed the prerequisite, you can prevent data loss during a failover of the DBFS connection after a failure of the back-end Oracle database instance. In this case, cached *writes* may be lost if the client loses the connection. However, back-end failover to other Oracle RAC instances or standby databases does not cause lost writes.

- Specify the `-o failover` mount option:

  ```
  $ dbfs_client database_user@database_server -o failover /mnt/dbfs
  ```

18.5.4 Bypassing Client-Side Write Caching

The sharing and caching semantics for `dbfs_client` are similar to NFS in using the *close-to-open cache consistency* behavior.

This allows multiple copies of `dbfs_client` to access the same shared file system. The default mode caches writes on the client and flushes them after a timeout or after the user closes the file. Also, writes to a file only appear to clients that open the file after the writer closed the file.

You can bypass client-side write caching.

- Specify `O_SYNC` when the file is opened.

  To force writes in the cache to disk call `fsync`.

18.5.5 Backing up DBFS

You have two alternatives for backing up DBFS.

You can back up the tables that underlie the file system at the database level or use a file system backup utility, such as Oracle Secure Backup, through a mount point.

Topics:

- DBFS Backup at the Database Level
  An advantage of backing up the tables at the database level is that the files in the file system are always consistent with the relational data in the database.
- DBFS Backup Through a File System Utility
  The advantage of backing up the file system using a file system backup utility is that individual files can be restored from backup more easily.

18.5.5.1 DBFS Backup at the Database Level

An advantage of backing up the tables at the database level is that the files in the file system are always consistent with the relational data in the database.

A full restore and recover of the database also fully restores and recovers the file system with no data loss. During a point-in-time recovery of the database, the files are recovered to the specified time. As usual with database backup, modifications that occur during the backup do not affect the consistency of a restore. The entire restored file system is always consistent with respect to a specified time stamp.

18.5.5.2 DBFS Backup Through a File System Utility

The advantage of backing up the file system using a file system backup utility is that individual files can be restored from backup more easily.

Any changes made to the restored files after the last backup are lost.

Specify the `allow_root` mount option if backups are scheduled using the Oracle Secure Backup Administrative Server.

18.5.6 Small File Performance of DBFS

Like any shared file system, the performance of DBFS for small files lags the performance of a local file system.

Each file data or metadata operation in DBFS must go through the `FUSE` user mode file system and then be forwarded across the network to the database. Therefore, each operation that is not cached on the client takes a few milliseconds to run in DBFS.

For operations that involve an input/output (IO) to disk, the time delay overhead is masked by the wait for the disk IO. Naturally, larger IOs have a lower percentage overhead than smaller IOs. The network overhead is more noticeable for operations that do not issue a disk IO.

When you compare the operations on a few small files with a local file system, the overhead is not noticeable, but operations that affect thousands of small files incur a much more noticeable overhead. For example, listing a single directory or looking at a single file produce near instantaneous response, while searching across a directory tree with many thousands of files results in a larger relative overhead. Oracle recommends `direct_io` option in `dbfs_client` for optimal performance for reads and writes.

18.6 Shrinking and Reorganizing DBFS Filesystems

DBFS uses Online File system Reorganization to shrink itself, enabling the release of allocated space back to the containing tablespace.

- About Changing DBFS File Systems
  DBFS file systems, like other database segments, grow dynamically with the addition or enlargement of files and directories.
- Advantages of Online Filesystem Reorganization
  DBFS Online Filesystem Reorganization is a powerful data movement facility with these certain advantages.
- Determining Availability of Online Filesystem Reorganization
  DBFS for Oracle Database 12*c* and later supports online filesystem reorganization. Some earlier versions also support the facility.
- Required Permissions for Online Filesystem Reorganization
  Database users must have the following set of privileges for Online Filesystem Reorganizaton.
- Invoking Online Filesystem Reorganization
  You can perform an Online Filesystem Reorganization by creating a temporary DBFS filesystem.

18.6.1 About Changing DBFS File Systems

DBFS file systems, like other database segments, grow dynamically with the addition or enlargement of files and directories.

Growth occurs with the allocation of space from the tablespace that holds the DBFS file system to the various segments that make up the file system.

However, even if files and directories in the DBFS file system are deleted, the allocated space is not released back to the containing tablespace, but continues to exist and be available for other DBFS entities. A process called Online Filesystem Reorganization solves this problem by shrinking the DBFS Filesystem.

The DBFS Online Filesystem Reorganization utility internally uses the Oracle Database online redefinition facility, with the original file system and a temporary placeholder corresponding to the base and interim objects in the online redefinition model.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for further information about online redefinition

18.6.2 Advantages of Online Filesystem Reorganization

DBFS Online Filesystem Reorganization is a powerful data movement facility with these certain advantages.

These are:

- **It is online:** When reorganization is taking place, the filesystem remains fully available for read and write operations for all applications.
- **It can reorganize the structure:** The underlying physical structure and organization of the DBFS filesystem can be changed in many ways, such as:
  - A non-partitioned filesystem can be converted to a partitioned filesystem and vice-versa.
  - Special SecureFiles LOB properties can be selectively enabled or disabled in any combination, including the compression, encryption, and deduplication properties.
  - The data in the filesystem can be moved across tablespaces or within the same tablespace.
- **It can reorganize multiple filesystems concurrently:** Multiple different filesystems can be reorganized at the same time, if no temporary filesystems have the same name and the tablespaces have enough free space, typically, twice the space requirement for each filesystem being reorganized.

18.6.3 Determining Availability of Online Filesystem Reorganization

DBFS for Oracle Database 12*c* and later supports online filesystem reorganization. Some earlier versions also support the facility.

To determine if your version does, query for a specific function in the DBFS PL/SQL packages, as shown below:

- Query for a specific function in the DBFS PL/SQL packages.

```
$ sqlplus / as sysdba
SELECT * FROM dba_procedures
WHERE   owner = 'SYS'
    and object_name = 'DBMS_DBFS_SFS'
    and procedure_name = 'REORGANIZEFS';
```

If this query returns a single row similar to the one in this output, the DBFS installation supports Online Filesystem Reorganization. If the query does not return any rows, then the DBFS installation should either be upgraded or requires a patch for bug-10051996.

```
OWNER
--------------------------------------------------------------------------------
OBJECT_NAME
--------------------------------------------------------------------------------
PROCEDURE_NAME
--------------------------------------------------------------------------------
OBJECT_ID|SUBPROGRAM_ID|OVERLOAD                          |OBJECT_TYPE  |AGG|PIP
----------|-------------|---------------------------------|-------------|---|---
IMPLTYPEOWNER
--------------------------------------------------------------------------------
IMPLTYPENAME
--------------------------------------------------------------------------------
PAR|INT|DET|AUTHID
---|---|---|------------
SYS
DBMS_DBFS_SFS
REORGANIZEFS
     11424|           52|(null)                           |PACKAGE      |NO |NO
(null)
(null)
NO |NO |NO |CURRENT_USER
```

18.6.4 Required Permissions for Online Filesystem Reorganization

Database users must have the following set of privileges for Online Filesystem Reorganizaton.

Users must have these privileges:

- ALTER ANY TABLE
- DROP ANY TABLE
- LOCK ANY TABLE
- CREATE ANY TABLE
- SELECT ANY TABLE
- REDEFINE ANY TABLE
- CREATE ANY TRIGGER
- CREATE ANY INDEX
- CREATE TABLE
- CREATE MATERIALIZED VIEW
- CREATE TRIGGER

18.6.5 Invoking Online Filesystem Reorganization

You can perform an Online Filesystem Reorganization by creating a temporary DBFS filesystem.

> **Note:**
>
> Ensure that you don't create the temporary DBFS filesystem in the SYS schema. DBFS Online Filesystem Reorganization will not work if you create the temporary DBFS filesystem in the SYS schema.

1. Create a temporary DBFS filesystem with the desired new organization and structure: including the desired target tablespace (which may be the same tablespace as the filesystem being reorganized), desired target SecureFiles LOB storage properties (compression, encryption, or deduplication), and so on.
2. Invoke the PL/SQL procedure to reorganize the DBFS filesystem using the newly-created temporary filesystem for data movement.
3. Once the reorganization procedure completes, drop the temporary filesystem.

The example below reorganizes DBFS filesystem `FS1` in tablespace `TS1` into a new tablespace `TS2`, using a temporary filesystem named `TMP_FS`, where all filesystems belong to database user `dbfs_user`:

```
$ cd $ORACLE_HOME/rdbms/admin
$ sqlplus dbfs_user/***

@dbfs_create_filesystem TS2 TMP_FS
EXEC DBMS_DBFS_SFS.REORGANIZEFS('FS1', 'TMP_FS');
@dbfs_drop_filesystem TMP_FS
QUIT;
```

where:

- `TMP_FS` can have any valid name. It is intended as a temporary placeholder and can be dropped (as shown in the example above) or retained as a fully materialized point-in-time snapshot of the original filesystem.
- `FS1` is the original filesystem and is unaffected by the attempted reorganization. It remains usable for all DBFS operations, including SQL, PL/SQL, and `dbfs_client` mounts and commandline, during the reorganization. At the end of the reorganization, FS1 has the new structure and organization used to create `TMP_FS` and vice versa (`TMP_FS` will have the structure and organization originally used for `FS1`). If the reorganization fails for any reason, DBFS attempts to clean up the internal state of `FS1`.
- `TS2` needs enough space to accommodate all active (non-deleted) files and directories in `FS1`.
- `TS1` needs at least twice the amount of space being used by `FS1` if the filesystem is moved within the same tablespace as part of a shrink.