

[Documentation](#) > [Databricks data engineering](#) > Work with files on Databricks

# Work with files on Databricks

August 09, 2024

Databricks has multiple utilities and APIs for interacting with files in the following locations:

- Unity Catalog volumes
- Workspace files
- Cloud object storage
- DBFS mounts and DBFS root
- Ephemeral storage attached to the driver node of the cluster

This article has examples for interacting with files in these locations for the following tools:

- Apache Spark
- Spark SQL and Databricks SQL
- Databricks file system utilities (`dbutils.fs` or `%fs`)
- Databricks CLI
- Databricks REST API
- Bash shell commands (`%sh`)

- Notebook-scoped library installs using %pip
- pandas
- OSS Python file management and processing utilities

### Important!

File operations requiring FUSE data access cannot directly access cloud object storage using URIs. Databricks recommends using Unity Catalog volumes to configure access to these locations for FUSE.

Scala supports FUSE for Unity Catalog volumes and workspace files on compute configured with Unity Catalog and shared access mode. On compute configured with single user access mode and Databricks Runtime 14.3 and above, Scala supports FUSE for Unity Catalog volumes and workspace files, except for subprocesses that originate from Scala, such as the Scala command `"cat /Volumes/path/to/file".!!.`

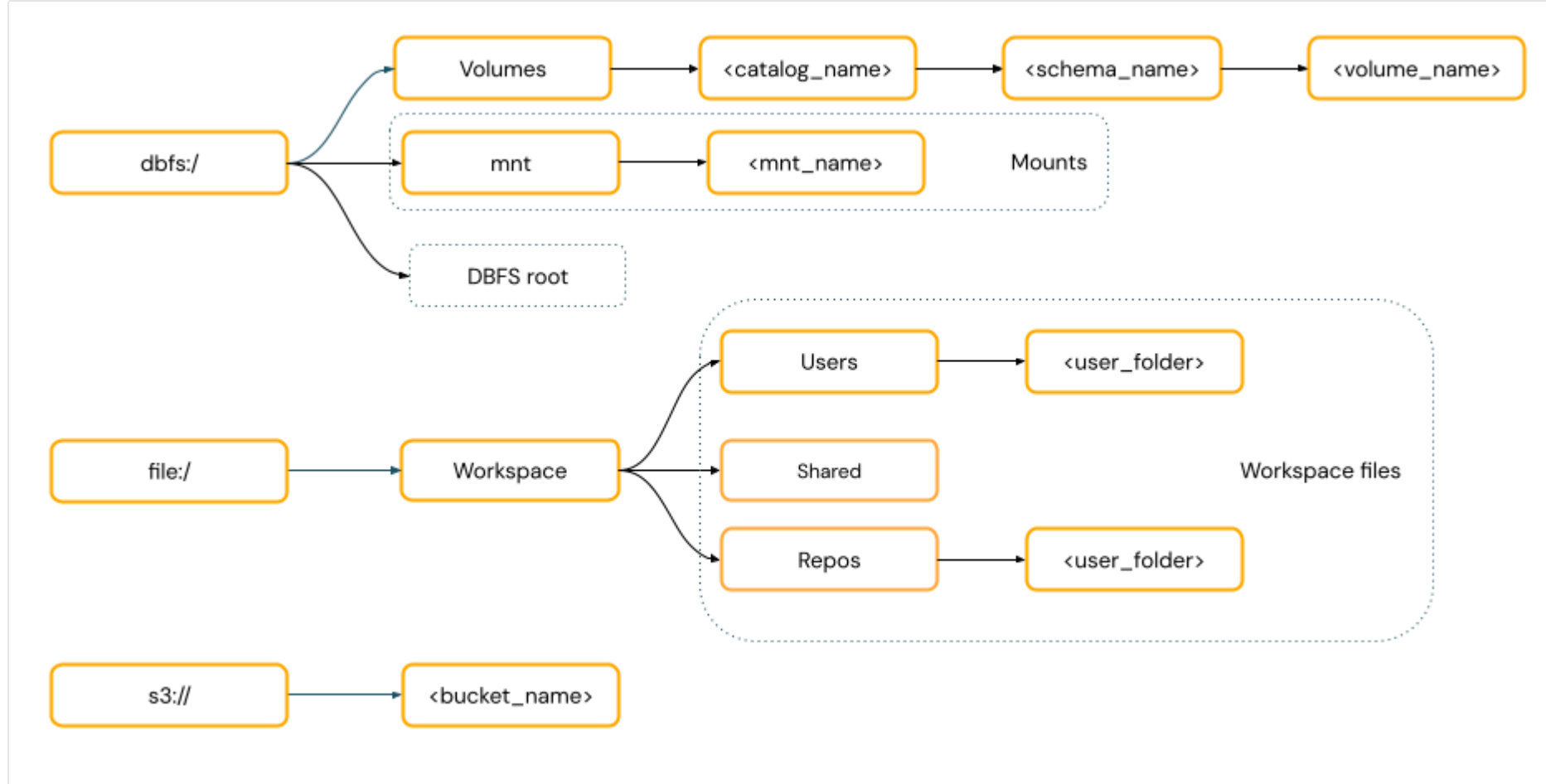
### In this article:

- [Do I need to provide a URI scheme to access data?](#)
- [Work with files in Unity Catalog volumes](#)
- [Work with workspace files](#)
- [Work with files in cloud object storage](#)
- [Work with files in DBFS mounts and DBFS root](#)
- [Work with files in ephemeral storage attached to the driver node](#)
- [Additional resources](#)

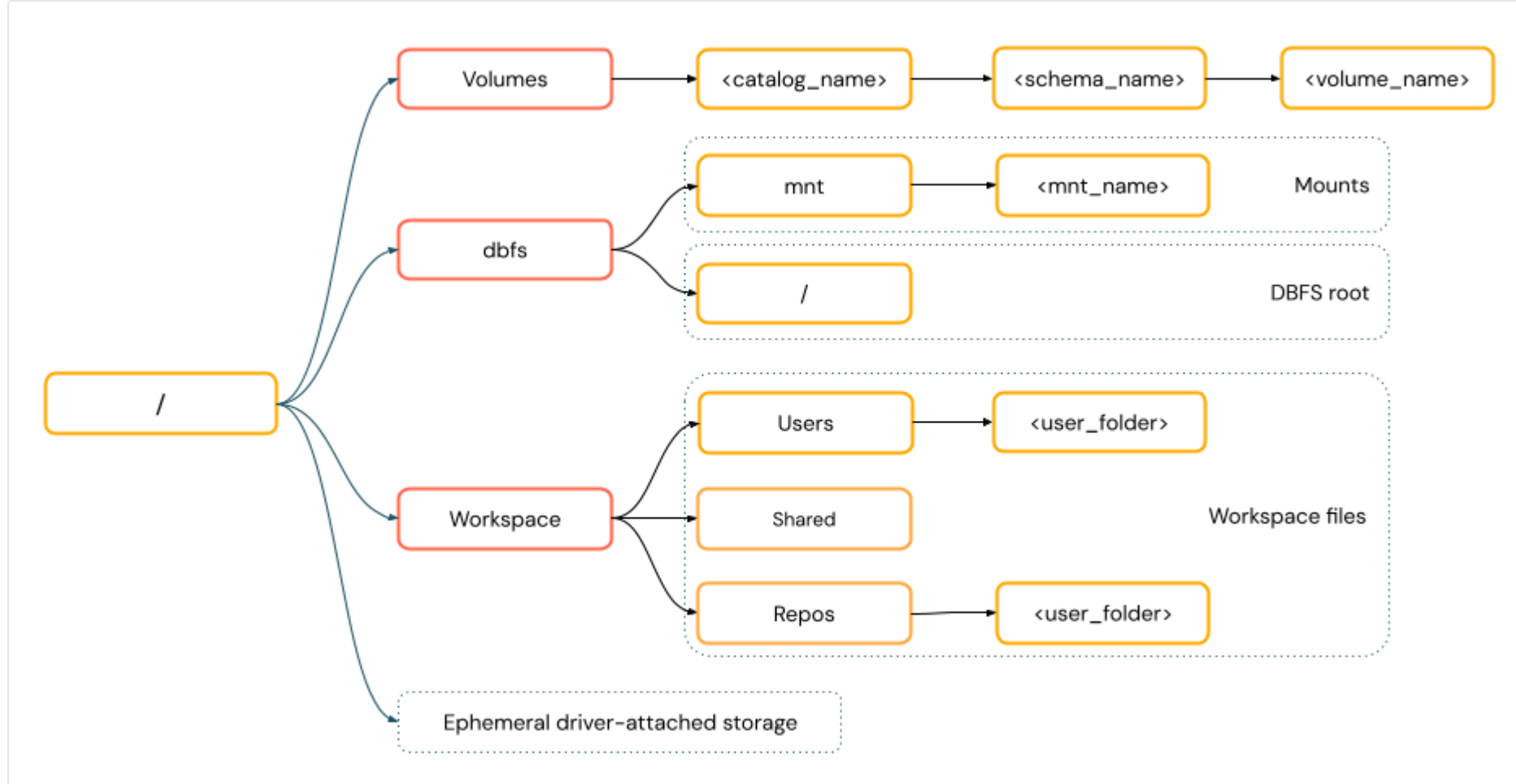
## Do I need to provide a URI scheme to access data?

Data access paths in Databricks follow one of the following standards:

- **URI-style paths** include a URI scheme. For Databricks-native data access solutions, URI schemes are optional for most use cases. When directly accessing data in cloud object storage, you must provide the correct URI scheme for the storage type.



- **POSIX-style paths** provide data access relative to the driver root (`/`). POSIX-style paths never require a scheme. You can use Unity Catalog volumes or DBFS mounts to provide POSIX-style access to data in cloud object storage. Many ML frameworks and other OSS Python modules require FUSE and can only use POSIX-style paths.



## Work with files in Unity Catalog volumes

Databricks recommends using Unity Catalog volumes to configure access to non-tabular data files stored in cloud object storage. See [What are Unity Catalog volumes?](#).

Tool	Example
Apache Spark	<code>spark.read.format("json").load("/Volumes/my_catalog/my_schema/my_volume/data.json").show()</code>

Tool	Example
Spark SQL and Databricks SQL	<code>SELECT * FROM csv.`/Volumes/my_catalog/my_schema/my_volume/data.csv`; LIST '/Volumes/my_catalog/my_schema/my_volume/</code>
Databricks file system utilities	<code>dbutils.fs.ls("/Volumes/my_catalog/my_schema/my_volume/") %fs ls /Volumes/my_catalog/my_schema/my_volume/</code>
Databricks CLI	<code>databricks fs cp /path/to/local/file dbfs:/Volumes/my_catalog/my_schema/my_volume/</code>
Databricks REST API	<code>POST https://&lt;databricks-instance&gt;/api/2.1/jobs/create {"name": "A multitask job", "tasks": [{..."libraries": [{"jar</code>
Bash shell commands	<code>%sh curl http://&lt;address&gt;/text.zip -o /Volumes/my_catalog/my_schema/my_volume/tmp/text.zip</code>
Library installs	<code>%pip install /Volumes/my_catalog/my_schema/my_volume/my_library.whl</code>
Pandas	<code>df = pd.read_csv('/Volumes/my_catalog/my_schema/my_volume/data.csv')</code>
OSS Python	<code>os.listdir('/Volumes/my_catalog/my_schema/my_volume/path/to/directory')</code>

## Note

The `dbfs:/` scheme is required when working with the Databricks CLI.

## Volumes limitations

Volumes have the following limitations:

- Direct-append or non-sequential (random) writes, such as writing Zip and Excel files are not supported. For direct-append or random-write workloads, perform the operations on a local disk first and then copy the results to Unity Catalog volumes. For example:

[Python](#)

 Copy

```
# python
import xlsxwriter
from shutil import copyfile

workbook = xlsxwriter.Workbook('/local_disk0/tmp/excel.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write(0, 0, "Key")
worksheet.write(0, 1, "Value")
workbook.close()

copyfile('/local_disk0/tmp/excel.xlsx', '/Volumes/my_catalog/my_schema/my_volume/excel.xlsx')
```

- Sparse files are not supported. To copy sparse files, use `cp --sparse=never`:

Bash

 Copy

```
$ cp sparse.file /Volumes/my_catalog/my_schema/my_volume/sparse.file
error writing '/dbfs/sparse.file': Operation not supported
$ cp --sparse=never sparse.file /Volumes/my_catalog/my_schema/my_volume/sparse.file
```

## Work with workspace files

Databricks [workspace files](#) are the files in a workspace that are not notebooks. You can use workspace files to store and access data and other files saved alongside notebooks and other workspace assets. Because workspace files have size restrictions, Databricks recommends only storing small data files here primarily for development and testing.

Tool	Example
Apache Spark	<code>spark.read.format("json").load("file:/Workspace/Users/&lt;user-folder&gt;/data.json").show()</code>
Spark SQL and Databricks SQL	<code>SELECT * FROM json.`file:/Workspace/Users/&lt;user-folder&gt;/file.json`;</code>

Tool	Example
Databricks file system utilities	<code>dbutils.fs.ls("file:/Workspace/Users/&lt;user-folder&gt;/") %fs ls file:/Workspace/Users/&lt;user-folder&gt;/</code>
Databricks CLI	<code>databricks workspace list</code>
Databricks REST API	<code>POST https://&lt;databricks-instance&gt;/api/2.0/workspace/delete {"path": "/Workspace/Shared/code.py", "recursive": "false"}</code>
Bash shell commands	<code>%sh curl http://&lt;address&gt;/text.zip -o /Workspace/Users/&lt;user-folder&gt;/text.zip</code>
Library installs	<code>%pip install /Workspace/Users/&lt;user-folder&gt;/my_library.whl</code>
Pandas	<code>df = pd.read_csv('/Workspace/Users/&lt;user-folder&gt;/data.csv')</code>
Scala	<code>val file = new File("/Workspace/Users/&lt;user-folder&gt;/data.csv")</code>

## Note

The `file:/` schema is required when working with Databricks Utilities, Apache Spark, or SQL.

# Workspace files limitations

Workspace files have the following limitations:

- Workspace file size is limited to 500MB. Operations that attempt to download or create files larger than this limit will fail.
- If your workflow uses source code located in a [remote Git repository](#), you cannot write to the current directory or write using a relative path. Write data to other location options.
- You cannot use `git` commands when you save to workspace files. The creation of `.git` directories is not allowed in workspace files.
- There is limited support for workspace file operations from **serverless compute**.

- Executors cannot write to workspace files.
- symlinks are not supported.
- Workspace files can't be accessed from [user-defined functions \(UDFs\)](#) on clusters with [shared access mode](#) on Databricks Runtime 14.2 and below.

# Where do deleted workspace files go?

Deleting a workspace file sends it to the trash. You can recover or permanently delete files from the trash using the UI.

See [Delete an object](#).

# Work with files in cloud object storage

Databricks recommends using Unity Catalog volumes to configure secure access to files in cloud object storage. You must configure permissions if you choose to directly access data in cloud object storage using URIs. See [Manage external locations, external tables, and external volumes](#).

The following examples use URIs to access data in cloud object storage:

Tool	Example
Apache Spark	<code>spark.read.format("json").load("s3://&lt;bucket&gt;/path/file.json").show()</code>
Spark SQL and Databricks SQL	<code>SELECT * FROM csv.`s3://&lt;bucket&gt;/path/file.json`; LIST 's3://&lt;bucket&gt;/path';</code>
Databricks file system utilities	<code>dbutils.fs.ls("s3://&lt;bucket&gt;/path/") %fs ls s3://&lt;bucket&gt;/path/</code>
Databricks CLI	Not supported
Databricks REST API	Not supported



Tool	Example
Bash shell commands	Not supported
Library installs	<code>%pip install s3://bucket-name/path/to/library.whl</code>
Pandas	Not supported
OSS Python	Not supported

### Note

Cloud object storage not support Amazon S3 mounts with client-side encryption enabled.

## Work with files in DBFS mounts and DBFS root

DBFS mounts are not securable using Unity Catalog and are no longer recommended by Databricks. Data stored in the DBFS root is accessible by all users in the workspace. Databricks recommends against storing any sensitive or production code or data in the DBFS root. See [What is DBFS?](#).

Tool	Example
Apache Spark	<code>spark.read.format("json").load("/mnt/path/to/data.json").show()</code>
Spark SQL and Databricks SQL	<code>SELECT * FROM json.`/mnt/path/to/data.json`;</code>
Databricks file system utilities	<code>dbutils.fs.ls("/mnt/path") %fs ls /mnt/path</code>
Databricks CLI	<code>databricks fs cp dbfs:/mnt/path/to/remote/file /path/to/local/file</code>

Tool	Example
Databricks REST API	<code>POST https://&lt;host&gt;/api/2.0/dbfs/delete --data '{ "path": "/tmp/HelloWorld.txt" }'</code>
Bash shell commands	<code>%sh curl http://&lt;address&gt;/text.zip &gt; /dbfs/mnt/tmp/text.zip</code>
Library installs	<code>%pip install /dbfs/mnt/path/to/my_library.whl</code>
Pandas	<code>df = pd.read_csv('/dbfs/mnt/path/to/data.csv')</code>
OSS Python	<code>os.listdir('/dbfs/mnt/path/to/directory')</code>

## Note

The `dbfs:/` scheme is required when working with the Databricks CLI.

# Work with files in ephemeral storage attached to the driver node

The ephemeral storage attached to the driver node is block storage with built-in POSIX-based path access. Any data stored in this location disappears when a cluster terminates or restarts.

Tool	Example
Apache Spark	Not supported
Spark SQL and Databricks SQL	Not supported
Databricks file system utilities	<code>dbutils.fs.ls("file:/path") %fs ls file:/path</code>

Tool	Example
Databricks CLI	Not supported
Databricks REST API	Not supported
Bash shell commands	<code>%sh curl http://&lt;address&gt;/text.zip &gt; /tmp/text.zip</code>
Library installs	Not supported
Pandas	<code>df = pd.read_csv('/path/to/data.csv')</code>
OSS Python	<code>os.listdir('/path/to/directory')</code>


## Note

The `file:/` schema is required when working with Databricks Utilities.

# Move data from ephemeral storage to volumes


You might want to access data downloaded or saved to ephemeral storage using Apache Spark. Because ephemeral storage is attached to the driver and Spark is a distributed processing engine, not all operations can directly access data here. Suppose you must move data from the driver filesystem to Unity Catalog volumes. In that case, you can copy files using [magic commands](#) or the [Databricks utilities](#), as in the following examples:

Python

 Copy

```
dbutils.fs.cp ("file:/<path>", "/Volumes/<catalog>/<schema>/<volume>/<path>")
```

Bash

 Copy

```
%sh cp /<path> /Volumes/<catalog>/<schema>/<volume>/<path>
```

Bash

 Copy

```
%fs cp file:/<path> /Volumes/<catalog>/<schema>/<volume>/<path>
```

## Additional resources

For information about uploading local files or downloading internet files to Databricks, see [Upload files to Databricks](#).

---

© Databricks 2024. All rights reserved. Apache, Apache Spark, Spark, and the Spark logo are trademarks of the [Apache Software Foundation](#).

[Send us feedback](#) | [Privacy Policy](#) | [Terms of Use](#)