

PUT

Uploads data files from a local file system to one of the following Snowflake stages:

- A named internal stage.
- A specified table's internal stage.
- The current user's internal stage.

You can load staged files into a table using the [COPY INTO <table>](#) command.

Note

- PUT does ***not*** support uploading files to external stages. To upload files to external stages, use the utilities provided by the cloud service.
- The [ODBC driver](#) supports PUT with Snowflake accounts hosted on the following platforms:
 - Amazon Web Services
 - Google Cloud Platform
 - Microsoft Azure

See also:

[GET](#) , [LIST](#) , [REMOVE](#) , [COPY FILES](#)

Syntax

```
PUT file://<path_to_file>/<filename> internalStage
[ PARALLEL = <integer> ]
[ AUTO_COMPRESS = TRUE | FALSE ]
[ SOURCE_COMPRESSION = AUTO_DETECT | GZIP | BZ2 | BROTLI | ZSTD |
DEFLATE | RAW_DEFLATE | NONE ]
[ OVERWRITE = TRUE | FALSE ]
```

Where:

```
internalStage ::=
    @[<namespace>.<int_stage_name>[/<path>]]
```

```
| @[<namespace>.%<table_name>[/<path>]]  
| @~[/<path>]
```

Required parameters

`file://<path_to_file>/<filename>`

Specifies the URI for the data files on the client machine, where:

- `<path_to_file>` is the local directory path to the files to upload.
- `<filename>` is the name of the file to upload. You can use wildcard characters (`*`, `?`) to upload multiple files. If the directory path or filename includes special characters or spaces, enclose the entire file URI in single quotes.

The URI formatting differs depending on your client operating system.

Linux/macOS You must include the initial forward slash in the path. For example, for a file named `load` use `file:///tmp/load`.

Windows You must include the drive and backslash in the path and replace backslash characters with forward slashes. For example, for a file named `load data` use `file://C:/temp/load data`.

Note

Snowflake doesn't support tar (tape archive) files.

internalStage

Specifies the location in Snowflake where to upload the files:

```
@[<namespace>.]  
<int_stage_name>[/<path>]
```

Files are uploaded to the specified named internal stage.

```
@[<namespace>.%<table_name>  
[/<path>]
```

Files are uploaded to the stage for the specified table.

```
@~[/<path>]
```

Files are uploaded to the stage for the current user.

Where:

- `<namespace>` is the database or schema that contains the named internal stage or table. It is **optional** if a database and schema are in use within the session.
- `<path>` is an optional case-sensitive path for files in the cloud storage location that limits access to a set of files. Paths are alternatively called *prefixes* or *folders* by different cloud storage services.

Note

If the stage name or path includes spaces or special characters, enclose it in single quotes. For example, use `'@"my stage"'` for a stage named `"my stage"`.

Optional parameters

PARALLEL = `<integer>`

Specifies the number of threads to use for uploading files. The upload process separate batches of data files by size:

- Small files (< 64 MB compressed or uncompressed) are staged in parallel as individual files.
- Larger files are automatically split into chunks, staged concurrently, and reassembled in the target stage. A single thread can upload multiple chunks.

Increasing the number of threads can improve performance when uploading large files.

Supported values: Any integer value from `1` (no parallelism) to `99` (use 99 threads for uploading files).

Default: `4`

Note

A 16 MB limit applies to older versions of Snowflake drivers, including:

- JDBC Driver versions prior to 3.12.1.
- ODBC Driver versions prior to 2.20.5.
- Python Connector versions prior to 2.2.0.

AUTO_COMPRESS = `TRUE` | `FALSE`

Specifies whether Snowflake uses gzip to compress files during upload:

- `TRUE`: Snowflake compresses the files (if they are not already compressed).
- `FALSE`: Snowflake doesn't compress the files.

This option does not support other compression types. To use a different compression type, compress the file separately before executing the PUT command. Then, identify the compression type using the `SOURCE_COMPRESSION` option.

Ensure your local folder has sufficient space for Snowflake to compress the data files before staging them. If necessary, set the `TEMP`, `TMPDIR` or `TMP` environment variable in your operating system to point to a local folder that contains additional free space.

Default: `TRUE`

`SOURCE_COMPRESSION = AUTO_DETECT | GZIP | BZ2 | BROTLI | ZSTD | DEFLATE | RAW_DEFLATE | NONE`

Specifies the method of compression used on already-compressed files that are being staged:

| Supported Values | Notes |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>AUTO_DETECT</code> | Compression algorithm detected automatically, except for Brotli-compressed files, which cannot currently be detected automatically. If loading Brotli-compressed files, explicitly use <code>BROTLI</code> instead of <code>AUTO_DETECT</code> . |
| <code>GZIP</code> | Doesn't support the <code>*.tar.gz</code> file format. |
| <code>BZ2</code> | Doesn't support the <code>*.tar.bz2</code> file format. |
| <code>BROTLI</code> | Must be used if loading Brotli-compressed files. |
| <code>ZSTD</code> | Zstandard v0.8 (and higher) supported. |
| <code>DEFLATE</code> | Deflate-compressed files (with zlib header, RFC1950). |
| <code>RAW_DEFLATE</code> | Raw Deflate-compressed files (without header, RFC1951). |
| <code>NONE</code> | Data files to load have not been compressed. |

Default: `AUTO_DETECT`

Note

Snowflake uses this option to detect how the data files were compressed so that they can be uncompressed and the data extracted for loading; it does ***not*** use this option to compress the files.

Uploading files that were compressed with other utilities is not currently supported.

OVERWRITE = TRUE | FALSE

Specifies whether Snowflake overwrites an existing file with the same name during upload:

- `TRUE`: An existing file with the same name is overwritten.
- `FALSE`: An existing file with the same name is not overwritten.

Note that a LIST operation on the stage is performed in the background, which can affect the performance of the PUT operation.

If attempts to PUT a file fail because a file with the same name exists in the target stage, the following options are available:

- Load the data from the existing file into one or more tables, and remove the file from the stage. Then PUT a file with new or updated data to the stage.
- Rename the local file, and then attempt the PUT operation again.
- Set `OVERWRITE = TRUE` in the PUT statement. Do this only if it is actually safe to overwrite a file with data that might not yet have been loaded into Snowflake.

Note that if your Snowflake account is hosted on Google Cloud Platform, PUT statements do not recognize when the OVERWRITE parameter is set to TRUE. A PUT operation ***always*** overwrites any existing files in the target stage with the local files you are uploading.


The following clients support the OVERWRITE option for Snowflake accounts hosted on Amazon Web Services or Microsoft Azure:

- SnowSQL
- Snowflake ODBC Driver
- Snowflake JDBC Driver
- Snowflake Connector for Python

Supported values: TRUE, FALSE.

Default: `FALSE`.

Usage notes

- The command cannot be executed from the **Worksheets**  page in either Snowflake web interface; instead, use the [SnowSQL client](#) or [Drivers](#) to upload data files, or check the documentation for a specific Snowflake client to verify support for this command.
- The command *does not* support uploading multiple files with divergent directory paths, because it doesn't preserve file system directory structure when uploading files to your stage.

For example, the following PUT statement returns an error since you can't specify multiple files in nested subdirectories.

```
PUT file:///tmp/data/** @my_int_stage AUTO_COMPRESS=FALSE;
```

- File-globbing patterns, like wildcards, are supported unless the files that match the pattern have divergent directory paths.
- The command does **not** create or rename files.
- All files stored on internal stages for data loading and unloading operations are automatically encrypted using AES-256 strong encryption on the server side. By default, Snowflake provides additional client-side encryption with a 128-bit key (with the option to configure a 256-bit key). For more information, see [encryption types for internal stages](#).
- The command ignores any duplicate files you attempt to upload to the same stage. A duplicate file is an unmodified file with the same name as an already-staged file.

To overwrite an already-staged file, you must modify the file you are uploading so that its contents are different from the staged file, which results in a new checksum for the newly-staged file.

- For the [PUT](#) and [GET](#) commands, an EXECUTION_STATUS of `success` in the [QUERY_HISTORY](#) does *not* mean that data files were successfully uploaded or downloaded. Instead, the status indicates that Snowflake received authorization to proceed with the file transfer.

Tip

For security reasons, the command times out after a set period of time. This can occur when loading large, uncompressed data files. To avoid timeout issues, we recommend compressing large data files using one of the supported compression types before uploading the files. Then, specify the compression type for the files using the `SOURCE_COMPRESSION` option.

You can also consider increasing the value of the `PARALLEL` option, which can help with performance when uploading large data files.

Furthermore, to take advantage of parallel operations when loading data into tables (using the [COPY INTO <table>](#) command), we recommend using data files ranging in size from

roughly 100 to 250 MB **compressed**. If your data files are larger, consider using a third-party tool to split them into smaller files before compressing and uploading them.

Examples

Upload a file named `mydata.csv` in the `/tmp/data` directory (in a Linux or macOS environment) to an internal stage named `my_int_stage`:

```
PUT file:///tmp/data/mydata.csv @my_int_stage;
```

Upload a file named `orders_001.csv` in the `/tmp/data` directory (in a Linux or macOS environment) to the stage for the `ordertiny_ext` table, with automatic data compression disabled:

```
PUT file:///tmp/data/orders_001.csv @ordertiny_ext AUTO_COMPRESS=FALSE;
```

Same example as above, but using wildcard characters in the filename to upload multiple files:

```
PUT file:///tmp/data/orders_*01.csv @ordertiny_ext AUTO_COMPRESS=FALSE;
```

Upload a file named `mydata.csv` in the `C:\temp\data` directory (in a Windows environment) to the stage for the current user, with automatic data compression enabled:

```
PUT file://C:/temp/data/mydata.csv @~ AUTO_COMPRESS=TRUE;
```

Similar to the previous example, but upload the file from the `C:\temp\load data` directory (in a Windows environment):

```
PUT 'file://C:/temp/load data/mydata.csv' @~ AUTO_COMPRESS=TRUE;
```