dbt\_project.yml

Every [dbt project](https://docs.getdbt.com/docs/build/projects) needs a dbt\_project.yml file — this is how dbt knows a directory is a dbt project. It also contains important information that tells dbt how to operate on your project.

By default, dbt will look for dbt\_project.yml in your current working directory and its parents, but you can set a different directory using the --project-dir flag or the DBT\_PROJECT\_DIR environment variable.

[name](https://docs.getdbt.com/reference/project-configs/name): string  
  
[config-version](https://docs.getdbt.com/reference/project-configs/config-version): 2  
[version](https://docs.getdbt.com/reference/project-configs/version): version  
  
[profile](https://docs.getdbt.com/reference/project-configs/profile): profilename  
  
[model-paths](https://docs.getdbt.com/reference/project-configs/model-paths): [directorypath]  
[seed-paths](https://docs.getdbt.com/reference/project-configs/seed-paths): [directorypath]  
[test-paths](https://docs.getdbt.com/reference/project-configs/test-paths): [directorypath]  
[analysis-paths](https://docs.getdbt.com/reference/project-configs/analysis-paths): [directorypath]  
[macro-paths](https://docs.getdbt.com/reference/project-configs/macro-paths): [directorypath]  
[snapshot-paths](https://docs.getdbt.com/reference/project-configs/snapshot-paths): [directorypath]  
[docs-paths](https://docs.getdbt.com/reference/project-configs/docs-paths): [directorypath]  
[asset-paths](https://docs.getdbt.com/reference/project-configs/asset-paths): [directorypath]  
  
[target-path](https://docs.getdbt.com/reference/project-configs/target-path): directorypath  
[log-path](https://docs.getdbt.com/reference/project-configs/log-path): directorypath  
[packages-install-path](https://docs.getdbt.com/reference/project-configs/packages-install-path): directorypath  
  
[clean-targets](https://docs.getdbt.com/reference/project-configs/clean-targets): [directorypath]  
  
[query-comment](https://docs.getdbt.com/reference/project-configs/query-comment): string  
  
[require-dbt-version](https://docs.getdbt.com/reference/project-configs/require-dbt-version): version-range | [version-range]  
  
[quoting](https://docs.getdbt.com/reference/project-configs/quoting):  
 database: true | false  
 schema: true | false  
 identifier: true | false  
  
models:  
 [<model-configs>](https://docs.getdbt.com/reference/model-configs)  
  
seeds:  
 [<seed-configs>](https://docs.getdbt.com/reference/seed-configs)  
  
snapshots:  
 [<snapshot-configs>](https://docs.getdbt.com/reference/snapshot-configs)  
  
sources:  
 [<source-configs>](https://docs.getdbt.com/reference/source-configs)  
   
tests:  
 [<test-configs>](https://docs.getdbt.com/reference/test-configs)  
  
vars:  
 [<variables>](https://docs.getdbt.com/docs/build/project-variables)  
  
[on-run-start](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end): sql-statement | [sql-statement]  
[on-run-end](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end): sql-statement | [sql-statement]  
  
[dispatch](https://docs.getdbt.com/reference/project-configs/dispatch-config):  
 - macro\_namespace: packagename  
 search\_order: [packagename]

1. .dbtignore

You can create a .dbtignore file in the root of your [dbt project](https://docs.getdbt.com/docs/build/projects) to specify files that should be **entirely** ignored by dbt. The file behaves like a [.gitignore file, using the same syntax](https://git-scm.com/docs/gitignore). Files and subdirectories matching the pattern will not be read, parsed, or otherwise detected by dbt—as if they didn't exist.

Syntax:

# .dbtignore  
  
# ignore individual .py files  
**not-a-dbt-model.py  
python\_dbt-model.py**  
  
# ignore all .py files  
**\*\*.py**  
# ignore all .py files with "codegen" in the filename  
***\*codegen\**.py**

We will have to simply put the file names under the .dbtignore syntax in dbt\_project.yml

1. analysis-paths

*dbt\_project.yml*

**analysis-paths:** [directorypath] – where all the analyses files resifes.

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/analysis-paths#definition)

We have to Specify a custom list of directories where [analyses](https://docs.getdbt.com/docs/build/analyses) files are located in the analysis\_path

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/analysis-paths#default)

*Without specifying this or (leaving it empty)* ***config (analysis\_path),*** *dbt will not compile any .sql files as analyses.*

However, the [dbt init command](https://docs.getdbt.com/reference/commands/init) populates this value as analyses -- **analysis-paths:** [“**analyses**”]

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/analysis-paths#examples)

* **Use a subdirectory named analyses**[**​**](https://docs.getdbt.com/reference/project-configs/analysis-paths#use-a-subdirectory-named-analyses)

dbt\_project.yml

**analysis-paths: ["analyses"]**

* **Use a subdirectory named custom\_analyses**[**​**](https://docs.getdbt.com/reference/project-configs/analysis-paths#use-a-subdirectory-named-custom_analyses)

dbt\_project.yml

**analysis-paths: ["custom\_analyses"]**

1. clean-targets

dbt\_project.yml

**clean-targets:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/clean-targets#definition)

Optionally specify a custom list of directories to be removed by the dbt clean [command](https://docs.getdbt.com/reference/commands/clean). As such, you should only include directories containing artifacts (e.g. compiled files, logs, installed packages) in this list.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/clean-targets#default)

If this configuration is not included in your dbt\_project.yml file, the clean command will remove files in your [target-path](https://docs.getdbt.com/reference/project-configs/target-path).

Examples[​](https://docs.getdbt.com/reference/project-configs/clean-targets#examples)

* **Remove packages and compiled files as part of dbt clean**[**​**](https://docs.getdbt.com/reference/project-configs/clean-targets#remove-packages-and-compiled-files-as-part-of-dbt-clean)

***This is our preferred configuration***

To remove packages as well as compiled files, include the value of your [packages-install-path](https://docs.getdbt.com/reference/project-configs/packages-install-path) configuration in your clean-targets configuration.

**dbt\_project.yml**

clean-targets: # directories to be removed by `dbt clean`

  - "target"

  - "dbt\_packages"

Now, ***run dbt clean.***

***Both the target and dbt\_packages directory will be removed.***

Note: this is the configuration in the dbt [starter project](https://github.com/dbt-labs/dbt-starter-project/blob/HEAD/dbt_project.yml), which is generated by the init command.

* **Remove logs when running dbt clean**[**​**](https://docs.getdbt.com/reference/project-configs/clean-targets#remove-logs-when-running-dbt-clean)

**dbt\_project.yml**

**clean-targets: [target, dbt\_packages, logs]**

1. config-version

**dbt\_project.yml**

**config-version: 2**

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/config-version#definition)

Specify your **dbt\_project.yml** as using the v2 structure.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/config-version#default)

Without this configuration, ***dbt will assume your dbt\_project.yml uses the version 1 syntax, which was deprecated in dbt v0.19.0.***

1. seed-paths

**dbt\_project.yml**

**seed-paths:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#definition)

Optionally specify a custom list of directories where [seed](https://docs.getdbt.com/docs/build/seeds) files are located.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#default)

By default, dbt expects seeds to be located in the seeds directory, i.e. **seed-paths: ["seeds"]**

Examples[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#examples)

* **Use a subdirectory named custom\_seeds instead of seeds**[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#use-a-subdirectory-named-custom_seeds-instead-of-seeds)

**dbt\_project.yml**

**seed-paths**: **["custom\_seeds"]**

* **Co-locate your models and seeds in the models directory**[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#co-locate-your-models-and-seeds-in-the-models-directory)

Note: this works because dbt is looking for different file types for seeds **(.csv files)** and models **(.sql files).**

**dbt\_project.yml**

**seed-paths:** **["models"]**  
**model-paths:** **["models"]**  
*This Is because seeds can only process .csv files and models only .sql so it will not put any effect if we keep seeds in models and give them same file locations*

* **Split your seeds across two directories**[**​**](https://docs.getdbt.com/reference/project-configs/seed-paths#split-your-seeds-across-two-directories)

Note: We recommend that you instead use two subdirectories within the seeds/ directory to achieve a similar effect.

**dbt\_project.yml**

**seed-paths:** **["seeds", "custom\_seeds"]**

*we can use two different sub directories inside the seeds paths and our seeds files will be split across these sub directories and it is the recommended option.*

1. docs-paths

**dbt\_project.yml**

**docs-paths:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/docs-paths#definition)

Optionally specify a custom list of directories where [**docs blocks**](https://docs.getdbt.com/docs/collaborate/documentation#docs-blocks) are located.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/docs-paths#default)

By default, dbt will search in all resource paths for docs blocks (i.e. the combined list of [model-paths](https://docs.getdbt.com/reference/project-configs/model-paths), [seed-paths](https://docs.getdbt.com/reference/project-configs/seed-paths), [analysis-paths](https://docs.getdbt.com/reference/project-configs/analysis-paths), [macro-paths](https://docs.getdbt.com/reference/project-configs/macro-paths) and [snapshot-paths](https://docs.getdbt.com/reference/project-configs/snapshot-paths)). If this option is configured, *dbt will only look in the specified directory for docs blocks.*

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/docs-paths#examples)

* **Use a subdirectory named docs for docs blocks**[**​**](https://docs.getdbt.com/reference/project-configs/docs-paths#use-a-subdirectory-named-docs-for-docs-blocks)

**dbt\_project.yml**

**docs-paths: ["docs"]**

INFO

**We typically omit this configuration as we prefer dbt's default behavior.**

**Run dbt docs generate — the assets directory will be copied to the target directory**

**Run dbt docs serve — the image will be rendered as part of your project documentation to the other users**

1. log-path

**dbt\_project.yml**

**log-path:** directorypath

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/log-path#definition)

Optionally specify a custom directory where dbt will write logs.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/log-path#default)

By default, dbt will write to the logs directory, i.e**. log-path: [“logs”]**

**Configuration**[**​**](https://docs.getdbt.com/reference/project-configs/log-path#configuration)

In the manner of a ["global" config](https://docs.getdbt.com/reference/global-configs/about-global-configs), the log path can be set in three places:

1. *--log-path CLI flag*
2. *DBT\_LOG\_PATH environment variable*
3. *log-path in dbt\_project.yml*

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/log-path#examples)

* **Write logs to a subdirectory named dbt\_logs instead of logs**[**​**](https://docs.getdbt.com/reference/project-configs/log-path#write-logs-to-a-subdirectory-named-dbt_logs-instead-of-logs)

**dbt\_project.yml**

**log-path:** **dbt\_logs**

FEATURE DEPRECATION

*As of dbt version 1.5, setting the log-path in the dbt\_project.yml is deprecated. Backward compatibility is still supported in 1.5 but will be removed in a future update. Migrate to the CLI flag or environment variable methods to avoid potential errors or disruptions.*

The precedence order is: ***CLI flag > env var > dbt\_project.yml(deprecated)***

1. macro-paths

**dbt\_project.yml**

**macro-paths:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/macro-paths#definition)

Optionally specify a custom list of directories where [macros](https://docs.getdbt.com/docs/build/jinja-macros#macros) are located. ***Note that you cannot co-locate models and macros.***

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/macro-paths#default)

By default, dbt will search for macros in a directory named macros, i.e. **macro-paths: ["macros"]**

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/macro-paths#examples)

* **Use a subdirectory named custom\_macros instead of macros**[**​**](https://docs.getdbt.com/reference/project-configs/macro-paths#use-a-subdirectory-named-custom_macros-instead-of-macros)

**dbt\_project.yml**

**macro-paths**: **["custom\_macros"]**

1. packages-install-path

**dbt\_project.yml**

**packages-install-path**: directorypath

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/packages-install-path#definition)

Optionally specify a custom directory where [packages](https://docs.getdbt.com/docs/build/packages) are installed when you run the dbt deps [command](https://docs.getdbt.com/reference/commands/deps). Note that this directory is usually git-ignored.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/packages-install-path#default)

By default, dbt will install packages in the **dbt\_packages** directory, i.e.

**packages-install-path**: **dbt\_packages**

Examples[​](https://docs.getdbt.com/reference/project-configs/packages-install-path#examples)

* **Install packages in a subdirectory named packages instead of dbt\_packages**[**​**](https://docs.getdbt.com/reference/project-configs/packages-install-path#install-packages-in-a-subdirectory-named-packages-instead-of-dbt_packages)

**dbt\_project.yml**

**packages-install-path:** **packages**

1. name

**dbt\_project.yml**

**name:** string (It is the name for the dbt\_project.yml)

Definition[**​**](https://docs.getdbt.com/reference/project-configs/name#definition)

**Required configuration**

The name of a dbt project. Must be letters, digits and underscores only, ***and cannot start with a digit.***

Recommendation[**​**](https://docs.getdbt.com/reference/project-configs/name#recommendation)

Often an organization has one dbt project, so it is sensible to name a project with your organization's name, in snake\_case. **For example:**

* name: acme
* name: jaffle\_shop
* name: evilcorp

Troubleshooting[​](https://docs.getdbt.com/reference/project-configs/name#troubleshooting)

**Invalid project name**[**​**](https://docs.getdbt.com/reference/project-configs/name#invalid-project-name)

*Encountered an error while reading the project:  
 ERROR: Runtime Error  
 at path ['name']: 'jaffle-shop' does not match '****^[^\\d\\W]\\w\*$****'  
Runtime Error  
 Could not run dbt*  
***(when we set the name of the project with digits, symbols, hyphens then It will throw such errors)***

* **This project has:**

**dbt\_project.yml**

**name:** jaffle-shop

*In this case, change your project name to be snake\_case instead:*

**dbt\_project.yml**

**name:** ***jaffle\_shop***

1. profile

**dbt\_project.yml**

**profile**: string (we can assign any name to the profiles that would be in strings)

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/profile#definition)

The profile is a file in dbt core which consists of connection settings of warehouse in your dbt project

* If you are developing in dbt Cloud: This configuration is not applicable
* If you are developing locally: This configuration is required, unless a command-line option (i.e. --profile) is supplied.

**Recommendation**[**​**](https://docs.getdbt.com/reference/project-configs/profile#recommendation)

Often an organization has only one [data warehouse](https://docs.getdbt.com/terms/data-warehouse), so it is sensible to use your organization's name as a profile name, in snake\_case. For example:

* profile: acme
* profile: jaffle\_shop

It is also reasonable to include the name of your warehouse technology in your profile name, particularly if you have multiple warehouses. For example:

* profile: acme\_snowflake
* profile: jaffle\_shop\_bigquery
* profile: jaffle\_shop\_redshift

1. require-dbt-version

**dbt\_project.yml**

**require-dbt-version:** version-range | [version-range]

Definition

***You can use require-dbt-version to restrict your project to only work with a range of dbt versions.***

*When you set this configuration, dbt sends a helpful error message for any user who attempts to run the project with an unsupported version of dbt*. This can be useful for package maintainers (such as dbt-utils) to ensure that users' dbt version is compatible with the package. Setting this configuration might also help your whole team remain synchronized on the same version of dbt for local development, to avoid compatibility issues from changed behaviour.

***If this configuration is not specified, no version check will occur.***

YAML QUOTING

This configuration needs to be interpolated by the YAML parser as a string. As such,***you should quote the value of the configuration, taking care to avoid whitespace***. For example:

* # ✅ **These will work**

**require-dbt-version:** **">=1.0.0"** # *Double quotes are OK*

**require-dbt-version:** **'>=1.0.0'** # *So are single quotes*

* # ❌ **These will not work**

**require-dbt-version:** **>=1.0.0** # No quotes? No good

**require-dbt-version:** **">= 1.0.0"** # Don't put whitespace after the equality signs

Examples

## Specify a minimum dbt version

Use a **>=** operator for a minimum boundary. In the following example, this project will run with any version of dbt greater than or equal to **1.0.0.**

**dbt\_project.yml**

**require-dbt-version:** **">=1.0.0"**

## ***Pin to a range***

Use a comma separated list for an upper and lower bound. In the following example, this project will run with dbt **1.x.x.**

**dbt\_project.yml**

**require-dbt-version: [">=1.0.0", "<2.0.0"]**

# OR

**dbt\_project.yml**

**require-dbt-version:** **">=1.0.0,<2.0.0"**

## ***Require a specific dbt version***

**NOT RECOMMENDED**

With the release of major version 1.0 of dbt Core, ***pinning to a specific patch is discouraged.***

While **you can restrict your project to run only with an exact version of dbt Core, we do not recommend** this for dbt Core v1.0.0 and higher.

# In the following example, the project will only run with dbt v0.21.1.

**dbt\_project.yml**

**require-dbt-version:** **0.21.1**

Invalid dbt versions

If the version of dbt used to invoke a project disagrees with the specified require-dbt-version in the project or any of the included packages, *then dbt will fail immediately with the following error:*

**$ dbt compile**

*Running with dbt=0.21.0*

*Encountered an error while reading the project:*

*Runtime Error*

*This version of dbt is not supported with the 'my\_project' package.*

*Installed version of dbt: =****0.21.0***

*Required version of dbt for 'my\_project'****: ['>=1.0.0', '<2.0.0']***

*Check the requirements for the 'my\_project' package, or run dbt again with --no-version-check*

Disabling version checks

To suppress failures to incompatible dbt versions, supply the **--no-version-check** flag to dbt run.

**$ dbt run --no-version-check**

*Running with dbt=0.21.0*

*Found 13 models, 2 tests, 1 archives, 0 analyses, 204 macros, 2 operations....*

1. snapshot-paths

**dbt\_project.yml**

**snapshot-paths:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/snapshot-paths#definition)

Optionally specify a custom list of directories where [snapshots](https://docs.getdbt.com/docs/build/snapshots) are located. Note that you cannot co-locate models and snapshots.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/snapshot-paths#default)

By default, dbt will search for snapshots in the snapshots directory, i.e.

**snapshot-paths:** **["snapshots"]**

Examples[​](https://docs.getdbt.com/reference/project-configs/snapshot-paths#examples)

**Use a subdirectory named archives instead of snapshots**[**​**](https://docs.getdbt.com/reference/project-configs/snapshot-paths#use-a-subdirectory-named-archives-instead-of-snapshots)

**dbt\_project.yml**

**snapshot-paths: ["archives"]**

1. model-paths

**dbt\_project.yml**

**model-paths**: [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/model-paths#definition)

Optionally specify a custom list of directories where [models](https://docs.getdbt.com/docs/build/models) and [sources](https://docs.getdbt.com/docs/build/sources) are located.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/model-paths#default)

By default, dbt will search for models and sources in the models directory,

i.e. **model-paths**: **["models"]**

Examples[​](https://docs.getdbt.com/reference/project-configs/model-paths#examples)

**Use a subdirectory named transformations instead of models**[**​**](https://docs.getdbt.com/reference/project-configs/model-paths#use-a-subdirectory-named-transformations-instead-of-models)

**dbt\_project.yml**

**model-paths:** **["transformations"]**

1. target-path

**dbt\_project.yml**

**target-path:** directorypath

**Definition**

Optionally specify a custom directory where compiled files (e.g. compiled models and tests) will be written when you run the dbt run, dbt compile, or dbt test command.

**Default**

By default, dbt will write compiled files to the target directory, i.e.

**target-path:** **target** # directory which will store compiled SQL files

Configuration

In the manner of a "global" config, the target path can be set in three places:

* **--target-path** CLI flag
* **DBT\_TARGET\_PATH** environment variable
* **target-path** in dbt\_project.yml

*FEATURE DEPRECATION*

*As of dbt version 1.5, setting the target-path in the dbt\_project.yml is deprecated. Backward compatibility is still supported in 1.5 but will be removed in a future update.* **Migrate to the CLI flag or environment variable methods to avoid potential errors or disruptions.**

Precedence:

The precedence order is: CLI flag > env var > dbt\_project.yml(deprecated)

**Examples**

Use a subdirectory named compiled for compiled files

**dbt\_project.yml**

**target-path:** **"compiled"**

1. test-paths

**dbt\_project.yml**

**test-paths:** [directorypath]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/test-paths#definition)

Optionally specify a custom list of directories where [singular tests](https://docs.getdbt.com/docs/build/tests) are located.

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/test-paths#default)

Without specifying this config, dbt will search for tests in the tests directory, i.e**. test-paths: ["tests"].** Specifically, it will look for .sql files containing:

* Generic test definitions in the tests/generic subdirectory
* Singular tests (all other files)

Examples[​](https://docs.getdbt.com/reference/project-configs/test-paths#examples)

**Use a subdirectory named custom\_tests instead of tests for data tests**[**​**](https://docs.getdbt.com/reference/project-configs/test-paths#use-a-subdirectory-named-custom_tests-instead-of-tests-for-data-tests)

**dbt\_project.yml**

**test-paths:** **["custom\_tests"]**

1. Configuring quoting in projects

**dbt\_project.yml**

**quoting:**  
 **database:** **true | false**  
 **schema:** **true | false**  
  **identifier: true | false**

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/quoting#definition)

Optionally configure ***whether*** dbt should quote databases, schemas, and identifiers when:

* creating relations (tables/views)
* resolving a ref function to a direct relation reference

***BIGQUERY TERMINOLOGY***

*Note that for BigQuery quoting configuration,****database and schema****should be used here, though these configs will apply to project and dataset names respectively*

**Default**[**​**](https://docs.getdbt.com/reference/project-configs/quoting#default)

The default values vary by database.

* Default
* Snowflake

Default

For most adapters, ***quoting is set to true by default.***

Why? It's equally easy to select from relations with quoted or unquoted identifiers. Quoting allows you to use reserved words and special characters in those identifiers, though we recommend avoiding this whenever possible.

**dbt\_project.yml**

**quoting:**  
 **database**: **true**  
 **schema**: **true**  
 **identifier**: **true**  
  
Snowflake

***On Snowflake, quoting is set to false by default.***

***Creating relations with quoted identifiers also makes those identifiers case sensitive. It's much more difficult to select from them***. You can re-enable quoting for relations identifiers that are case sensitive, reserved words, or contain special characters, but we recommend you avoid this as much as possible.

**dbt\_project.yml**

**quoting:**  
 **database**: **false**  
 **schema**: **false**  
 **identifier**: **false**

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/quoting#examples)

***Set quoting to false for a project:***

**dbt\_project.yml**

q**uoting:**  
 database: false  
 schema: false  
 identifier: false

* **dbt will then create relations without quotes if quoting is false:**

***create table analytics.dbt\_alice.dim\_customers***

Recommendation[**​**](https://docs.getdbt.com/reference/project-configs/quoting#recommendation)

Snowflake[**​**](https://docs.getdbt.com/reference/project-configs/quoting#snowflake)

Set all quoting configs to **False**. *This means that you cannot use reserved words as identifiers, however it's usually a good idea to avoid these reserved words anyway.*

If a Snowflake source table uses a quoted database, schema, or table identifier, you can configure it in the source.yml file. [Refer to configuring quoting for more info](https://docs.getdbt.com/reference/resource-properties/quoting).

**Explanation:**[**​**](https://docs.getdbt.com/reference/project-configs/quoting#explanation)

Whereas most databases will *lowercase* unquoted identifiers, **Snowflake will *uppercase* unquoted identifiers.** If a model name is lowercased *and quoted*, then it cannot be referred to without quotes! Check out the example below for more information.

**snowflake\_casing.sql**

*/\**  
 *You can run the following queries against your database*  
 *to build an intuition for how quoting works on Snowflake.*  
*\*/*  
  
*-- This is the output of an example `orders.sql` model with quoting enabled*

**create table "analytics"."orders" as (  
  
 select 1 as id  
  
);**  
  
*/\**  
 ***These queries WILL NOT work****! Since the table above was created with quotes,*  
 *Snowflake created the orders table with a lowercase schema and identifier.*  
  
 *Since unquoted identifiers are automatically uppercased, both of the*  
 *following queries are equivalent, and neither will work correctly.*  
*\*/*  
  
**select \* from analytics.orders;  
select \* from ANALYTICS.ORDERS;**  
  
*/\**  
 *To query this table, you'll need to quote the schema and table. This*  
 *query should indeed complete without error.*  
*\*/*  
  
**select \* from "analytics"."orders";**  
  
  
*/\**  
 ***To avoid this quoting madness, you can disable quoting for schemas*  
 *and identifiers in your dbt\_project.yml file.*** *This means that you*  
 *won't be able to use reserved words as model names, but you probably*  
 *shouldn't be doing that anyway! Assuming schema and identifier quoting is*  
 *disabled, the following query would indeed work:*  
*\*/*  
  
**select \* from analytics.orders;**

Another example:

select  
 ...  
  
*-- this should be quoted*  
from {{ source**('jaffle\_shop', 'orders'**) }}   
*-- here, the identifier should be unquoted*  
left join {{ source**('jaffle\_shop', 'customers'**) }} using ***(order\_id)***

conclusion

*we use quoting for a reason to assign those values to the database, schema and identifier in which there are symbols, digits or any irrelevant sign which will not going to be accepted like other normal names, for that reason we use quoting and first of all we enable the quoting in dbt\_project.yml for database, schema and identifier because as default they are marked false and we will have to make it true.*

*as when we have assigned the values in quotes then it will be called in the quotes and not directly without quotes as it will throw errors in it. Those quoted names are to be called in the quotes*

*select \* from {{ source****('jaffle\_shop', 'orders'****) }}*

1. on-run-start & on-run-end

**dbt\_project.yml**

**on-run-start**: sql-statement | [sql-statement]  
**on-run-end**: sql-statement | [sql-statement]

**Definition**[**​**](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end#definition)

A SQL statement (or list of SQL statements) to be run at the start, or end, of the following commands:

* *d bt run*
* *dbt test*
* *dbt seed*
* *dbt snapshot*
* *dbt build*
* *dbt compile*
* *dbt docs generate*

on-run-start and on-run-end hooks can also call macros that return SQL statements

**Usage notes**[**​**](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end#usage-notes)

* The on-run-end hook has additional jinja variables available in the context — check out the [docs](https://docs.getdbt.com/reference/dbt-jinja-functions/on-run-end-context).

**Examples**[**​**](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end#examples)

In older versions of dbt, the most common use of post-hook was to execute grant statements, to apply database permissions to models right after creating them. Starting in v1.2, we recommend using the [grants resource config](https://docs.getdbt.com/reference/resource-configs/grants) instead, in order to automatically apply grants when your dbt model runs.

**Grant privileges on all schemas that dbt uses at the end of a run**[**​**](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end#grant-privileges-on-all-schemas-that-dbt-uses-at-the-end-of-a-run)

This leverages the [schemas](https://docs.getdbt.com/reference/dbt-jinja-functions/schemas) variable that is only available in an on-run-end hook and grants it some privileges.

**dbt\_project.yml**

**on-run-end:**  
 **- "{% for schema in schemas %}**

grant usage on schema **{{ schema }}** to group reporter;

* **{% endfor %}"**In practice, it might not be a bad idea to put this code into a macro:

macros/grants.sql  
{% macro **grant\_usage\_to\_schemas**(schemas, user) %}  
 **{% for schema in schemas %}** grant usage on schema **{{ schema }}** to **{{ user }}**;  
 **{% endfor %}**  
**{% endmacro %}**

**Call a macro to grant privileges**[**​**](https://docs.getdbt.com/reference/project-configs/on-run-start-on-run-end#call-a-macro-to-grant-privileges)

**dbt\_project.yml**

on-run-end: "{{ grant\_select(schemas) }}"

* *now, after defining the macro we can only call the macro in the function on-run-end and pass the schema name and it will grant it all the privileges*

## database\_schemas[​](https://docs.getdbt.com/reference/dbt-jinja-functions/on-run-end-context#database_schemas)

The database\_schemas context variable can be used to reference the databases and schemas that dbt has built models into during a run of dbt. This variable is similar to the schemas variable, and should be used if a dbt run builds resources into multiple different databases.

Example:

macros/grants.sql

{% macro grant\_usage\_to\_schemas(database\_schemas, user) %}  
 {% for (database, schema) in database\_schemas %}  
 grant usage on {{ database }}.{{ schema }} to {{ user }};  
 {% endfor %}  
{% endmacro %}

dbt\_project.yml

on-run-end:  
 - "{{ grant\_usage\_to\_schemas(database\_schemas, user) }}"

1. Asset-paths:

In the context of the **asset-paths** configuration in **dbt\_project.yml**, it is specifically related to the **docs generate** command and the generation of project documentation.

When you specify the **asset-paths** configuration with a list of directory paths in **dbt\_project.yml**, it allows you to include additional directories that contain assets, such as images or other files, that you want to be copied to the target directory during the documentation generation process.

For example, if you have an **assets** directory in your project and you include it in the **asset-paths** configuration like this:

Sytax:

**dbt\_project.yml**

**config-version: 2**

**asset-paths:** - **"assets"**

When you run the **dbt docs generate** command, any files included in the specified **assets** directory will be copied to the **target/** directory along with the generated documentation. This allows you to include images or other files in your project's documentation, which can be referenced and displayed in the generated documentation.

By leveraging the **asset-paths** configuration, you can enhance your project's documentation by including additional assets that provide more context or visual representation alongside the generated documentation.

# dispatch (config)

**dbt\_project.yml**

**dispatch:**

- **macro\_namespace:** **packagename**

**search\_order:** **[packagename]**

- **macro\_namespace**: **packagename**

**search\_order:** **[packagename]**

In dbt, "dispatch" refers to the order in which the system looks for certain code (called "macros") that perform specific actions. The **dispatch** configuration in the **dbt\_project.yml** file lets you change the default order of searching for these macros in specific areas.

Imagine you have a package called **dbt\_utils** with macros that perform useful tasks. However, you want to use different versions of some of those macros, or even your own versions. This is where "dispatch" comes into play.

When you configure the **dispatch** setting, you're telling dbt to search for macros in specific places in a particular order. In the example given:

**dispatch:**

**- macro\_namespace:** **dbt\_utils**

**search\_order**: **['spark\_utils', 'dbt\_utils']**

It means that when looking for macros in the **dbt\_utils** namespace, dbt will first search in the **spark\_utils** package. If it doesn't find the macro there, it will then fallback to the versions in the original **dbt\_utils** package.

Now, what does "shim" mean in this context? A "shim" is a small piece of code that acts as a bridge between different software components. In this case, when you want to "shim" the **dbt\_utils** package with the **spark\_utils** package, you're essentially replacing or extending certain macros from the original package with your own versions or alternative implementations. By defining the search order explicitly, you're ensuring that your versions take priority over the default versions from **dbt\_utils**.

In summary, "dispatch" allows you to control the order of searching for macros in specific namespaces, giving you the flexibility to replace or augment macros from a package with your own versions or alternative implementations, referred to as "shimming."

Default version:

I've reimplemented certain macros from the dbt\_utils package in my root project **('my\_root\_project'),** and I want my versions to take precedence. Otherwise, fall back to the versions in dbt\_utils.

*Note: This is the default behavior. You may optionally choose to express that search order explicitly as:*

**dbt\_project.yml**

**dispatch:**  
 **- macro\_namespace: dbt\_utils**  
 **search\_order: ['my\_root\_project', 'dbt\_utils']**

(Now dbt will first search into the **'my\_root\_project',** and try to find the required macros in this and if it is not found inside it, dbt will then find the macros inside the **dbt\_utils**)

Conclusion:

Dispatch Is used to set the order for the dbt to search the macros/code in files or packages which is required, Imagine you have a package called **dbt\_utils** with macros that perform useful tasks. However, you want to use different versions of some of those macros, or even your own versions

Now if we say that the order is

**- macro\_namespace:** **dbt\_utils**

**search\_order**: **['spark\_utils', 'dbt\_utils'**

macro\_namespace is the name of the package or file we want to search and is intended to be used

search\_order is used to define the order in which searches are to be made. First spark\_utils and then dbt\_utils are to be searched

we can make the search order as we want like we want to search firstly in the root files then

**search\_order: ['my\_root\_project', 'dbt\_utils']**