Source: README.md

Dremio CLI Documentation

Complete documentation for the Dremio Command Line Interface.

Table of Contents

Getting Started

Core Operations

Data Management

Collaboration & Governance

Administration

Quick Start

```
# Install
pip install dremio-cli
```

```
# Configure profile
dremio profile create --name myprofile --type software \
  --base-url https://dremio.company.com \
  --username admin --password secret

# List catalog
dremio catalog list

# Execute SQL
dremio sql execute "SELECT * FROM customers LIMIT 10"

# Create a view
dremio view create --path "Analytics.customer_summary" \
  --sql "SELECT id, name, email FROM customers"
```

Documentation Guide

By Use Case

Data Exploration:

- Catalog - Browse available data
- SQL - Query your data
- Jobs - Monitor query execution

Data Engineering:

- Sources - Connect to data systems
- Tables - Configure physical datasets
- Views - Create virtual datasets

Data Governance:

- Tags & Wiki - Document datasets
- Grants - Control access
- Users & Roles - Manage users

Organization:

- Spaces & Folders - Structure your catalog
- Tags & Wiki - Categorize and document

Command Reference

Catalog Operations

```
dremio catalog list              # List catalog items
dremio catalog get <id>          # Get item details
dremio catalog get-by-path <path> # Get by path
```

SQL Operations

```
dremio sql execute <query>       # Execute SQL
dremio sql explain <query>       # Show execution plan
dremio sql validate <query>      # Validate syntax
```

Source Management

```
dremio source list               # List sources
dremio source create             # Create source
dremio source refresh <id>       # Refresh metadata
```

View Management

```
dremio view list                 # List views
dremio view create               # Create view
dremio view update <id>          # Update view
```

Job Management

```
dremio job list                  # List jobs
dremio job get <id>              # Get job details
dremio job results <id>          # Get results
dremio job cancel <id>           # Cancel job
```

Space & Folder Management

```
dremio space create --name <name>  # Create space
dremio folder create --path <path> # Create folder
```

Access Control

```
dremio grant list <id>           # List grants
dremio grant add <id>            # Add grant
dremio user list                 # List users
dremio role list                 # List roles
```

Platform Support

Feature	Software	Cloud
Catalog Operations	✓	✓
SQL Execution	✓	⚠ Limited
Job Management	✓	✓
View Management	✓	✓
Source Management	✓	✓
Space/Folder Management	✓	✓
Tags & Wiki	✓	✓
Grant Management	✓	✓
User Management	✓	⚠ Via Console
Role Management	✓	⚠ Via Console
Table Operations	✓	✓

Tips & Best Practices

Use profiles - Configure multiple profiles for different environments

JSON output - Use `output json` for scripting

Verbose mode - Add `verbose` for debugging

File-based operations - Store SQL queries and configs in files

Async execution - Use `async` for long-running queries

Examples

Data Pipeline

```
# 1. Create source
dremio source create --name MyDB --type POSTGRES --config-file db.json

# 2. Create space
dremio space create --name Analytics

# 3. Create view
dremio view create --path "Analytics.sales_summary" \
  --sql "SELECT date, SUM(amount) FROM sales GROUP BY date"

# 4. Grant access
dremio grant add <view-id> --grantee-type ROLE \
  --grantee-id analyst --privileges SELECT
```

Monitoring

```
# List recent jobs
dremio job list --max-results 10
```

```
# Get job details
dremio job get <job-id>

# Download profile
dremio job profile <job-id> --download profile.zip
```

Documentation

```
# Add wiki
dremio wiki set <id> --file README.md

# Add tags
dremio tag set <id> --tags "production,sensitive,pii"
```

Additional Resources

Dremio Documentation

Dremio Cloud API Reference

Dremio Software API Reference

Getting Help

```
# General help
dremio --help

# Command help
dremio <command> --help

# Subcommand help
dremio <command> <subcommand> --help
```

License

See LICENSE file for details.

Source: advanced-features.md

Advanced Features

This guide covers advanced CLI features for power users.

Query History

The CLI automatically tracks your query execution history in a local SQLite database.

List History

```
dremio history list
dremio history list --limit 10
```

Re-run from History

```
# List history to find ID
dremio history list

# Re-run command
dremio history run 5
```

Clear History

```
dremio history clear
```

Storage Location: `~/.dremio/history.db`

Favorite Queries

Save frequently used queries as favorites for quick access.

Add Favorite

```
dremio favorite add daily_report --sql "SELECT * FROM sales WHERE date = CURRENT_DATE"

dremio favorite add customer_count --sql "SELECT COUNT(*) FROM customers" \
  --description "Total customer count"
```

List Favorites

```
dremio favorite list
```

Run Favorite

```
dremio favorite run daily_report
```

Delete Favorite

```
dremio favorite delete daily_report
```

Interactive Mode

Launch an interactive REPL for executing multiple commands.

```
dremio repl
```

Features:

- Execute commands interactively
- Built-in help system
- Command history (up/down arrows)
- Exit with `exit`, `quit`, or Ctrl+D

Built-in Commands:

- `help` - Show available commands
- `help <command>` - Show detailed help for specific command
- `exit` or `quit` - Exit REPL

Example Session:

```
$ dremio repl
Dremio CLI - Interactive Mode
Type 'help' for available commands, 'exit' or 'quit' to exit.

Using profile: default

dremio> help
┌──────────────────┬──────────────────────────────┐
│ Command          │ Description                  │
├──────────────────┼──────────────────────────────┤
│ catalog          │ Browse and navigate catalog  │
│ sql              │ Execute SQL queries          │
│ job              │ Manage jobs                  │
│ view             │ Manage views                 │
│ source           │ Manage sources               │
│ space            │ Manage spaces                │
│ folder           │ Manage folders               │
│ grant            │ Manage permissions           │
│ history          │ View command history         │
│ favorite         │ Manage favorite queries      │
│ help [command]   │ Show help for command        │
│ exit/quit        │ Exit REPL                    │
└──────────────────┴──────────────────────────────┘

Examples:
```

```
  catalog list
  sql execute "SELECT * FROM table LIMIT 10"
  help sql

dremio> help sql
Usage: dremio sql [OPTIONS] COMMAND [ARGS]...

  SQL operations.

Options:
  --help  Show this message and exit.

Commands:
  execute   Execute a SQL query.
  explain   Explain a SQL query.
  validate  Validate SQL syntax.

dremio> catalog list
┌──────────────┬────────────┬──────────┐
│ Path         │ Type       │ ID       │
├──────────────┼────────────┼──────────┤
│ Analytics    │ SPACE      │ abc-123  │
│ MySource     │ SOURCE     │ def-456  │
└──────────────┴────────────┴──────────┘

dremio> exit
Goodbye!
```

Shell Auto-Completion

Enable Tab completion for commands and options.

Bash

```
# Install completion
source <(cat completions/dremio-completion.bash)

# Or add to ~/.bashrc
echo 'source /path/to/dremio-cli/completions/dremio-completion.bash' >> ~/.bashrc
```

Zsh

```
# Install completion
source completions/dremio-completion.zsh

# Or add to ~/.zshrc
echo 'source /path/to/dremio-cli/completions/dremio-completion.zsh' >> ~/.zshrc
```

Usage

```
# Tab completion for commands
dremio <TAB>
catalog  profile  source  space  ...

# Tab completion for subcommands
dremio catalog <TAB>
list  get  get-by-path

# Tab completion for options
dremio --<TAB>
--profile  --output  --verbose  --help
```

Workflows

Daily Reporting Workflow

```
# 1. Save daily report as favorite
dremio favorite add daily_sales --sql "
SELECT
  date,
  SUM(amount) as total_sales,
  COUNT(*) as transaction_count
FROM sales
WHERE date = CURRENT_DATE
GROUP BY date
"

# 2. Run daily
dremio favorite run daily_sales

# 3. Check history
dremio history list --limit 5
```

Interactive Exploration

```
# Launch REPL
dremio repl

# Explore catalog
dremio> catalog list
dremio> catalog get-by-path "Analytics.sales"

# Execute queries
dremio> sql execute "SELECT * FROM Analytics.sales LIMIT 10"

# Save useful query
dremio>  favorite  add  sales_summary  --sql  "SELECT  region,  SUM(amount)  FROM
```

```
Analytics.sales GROUP BY region"
```

Batch Operations with History

```
# Execute multiple queries
dremio sql execute "SELECT COUNT(*) FROM table1"
dremio sql execute "SELECT COUNT(*) FROM table2"
dremio sql execute "SELECT COUNT(*) FROM table3"

# Review history
dremio history list

# Re-run if needed
dremio history run 2
```

Tips

Use favorites for complex queries - Save time on frequently used queries

```
dremio favorite add monthly_report --sql "$(cat report.sql)"
```

History for debugging - Review past commands when troubleshooting

```
dremio history list --limit 20
```

REPL for exploration - Use interactive mode when learning the API

```
dremio repl
```

Completion for speed - Enable shell completion to type faster

```
source completions/dremio-completion.bash
```

Combine with pipes - Use standard Unix tools

```
dremio history list --output json | jq '.[] | select(.success == 1)'
```

Configuration

History Database Location

Default: `~/.dremio/history.db`

To use a different location, set the `DREMIO_HISTORY_DB` environment variable:

```
export DREMIO_HISTORY_DB=/custom/path/history.db
```

History Retention

History is stored indefinitely. Clear periodically:

```
# Clear all history
dremio history clear

# Or manually delete database
rm ~/.dremio/history.db
```

Summary

History - Automatic tracking of all commands

Favorites - Save and reuse common queries

REPL - Interactive command execution

Completion - Tab completion for faster typing

Source: catalog.md

Catalog Operations

This guide covers catalog operations including listing, retrieving, and navigating the Dremio catalog.

Commands

List Catalog

List all items in the catalog.

```
dremio catalog list [OPTIONS]
```

Options:

`include TEXT` - Include additional fields (e.g., `permissions`, `datasetCount`)

Examples:

```
# List all catalog items
dremio catalog list

# List with permissions
```

```
dremio catalog list --include permissions

# List with dataset count
dremio catalog list --include datasetCount

# JSON output
dremio --output json catalog list

# Use specific profile
dremio --profile software catalog list
```

Get Catalog Item by ID

Retrieve a specific catalog item by its ID.

```
dremio catalog get <ITEM_ID> [OPTIONS]
```

Arguments:

`ITEM_ID` - The catalog item ID (UUID)

Options:

`include TEXT` - Include additional fields

Examples:

```
# Get catalog item
dremio catalog get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3

# Get with SQL definition
dremio catalog get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --include sql

# Get with permissions
dremio catalog get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --include permissions

# YAML output
dremio --output yaml catalog get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3
```

Get Catalog Item by Path

Retrieve a catalog item by its path.

```
dremio catalog get-by-path <PATH> [OPTIONS]
```

Arguments:

`PATH` - The catalog path (dot-separated or slash-separated)

Options:

`include TEXT` - Include additional fields

Examples:

```
# Get by dot-separated path
dremio catalog get-by-path "MySpace.MyTable"

# Get by slash-separated path
dremio catalog get-by-path "MySpace/MyFolder/MyView"

# Cloud: source.namespace.object
dremio catalog get-by-path "evangelism-2026.testing.my_table"

# Software: space.object or catalog.namespace.object
dremio catalog get-by-path "Analytics.sales_data"
dremio catalog get-by-path "dremio-catalog.alexmerced.testing"

# With additional fields
dremio catalog get-by-path "MySpace.MyView" --include sql
```

Scenarios

Exploring the Catalog

```
# 1. List all top-level items
dremio catalog list

# 2. Find a specific space or source
dremio catalog list | grep "MySpace"

# 3. Get details about a space
dremio catalog get-by-path "MySpace"

# 4. Explore nested items
dremio catalog get-by-path "MySpace/Reports"
```

Finding Datasets

```
# List all items with dataset counts
dremio catalog list --include datasetCount

# Get specific dataset
dremio catalog get-by-path "Sales.customers"

# Check dataset permissions
dremio catalog get-by-path "Sales.customers" --include permissions
```

Working with Views

```
# Get view definition
dremio catalog get-by-path "Analytics.monthly_summary" --include sql

# Get view metadata
dremio --output json catalog get-by-path "Analytics.monthly_summary"
```

Cross-Environment Comparison

```
# Compare catalog between environments
dremio --profile dev catalog list > dev_catalog.json
dremio --profile prod catalog list > prod_catalog.json
diff dev_catalog.json prod_catalog.json
```

Output Formats

Table (Default)

```
dremio catalog list
```

Output:

```
┌──────────────────────┬──────┬──────────────┬─────────────┐
│ ID                   │ Path │ Type         │ Created     │
├──────────────────────┼──────┼──────────────┼─────────────┤
│ abc-123-def-456      │ ...  │ SPACE        │ 2024-...    │
│ xyz-789-ghi-012      │ ...  │ SOURCE       │ 2024-...    │
└──────────────────────┴──────┴──────────────┴─────────────┘
```

JSON

```
dremio --output json catalog list
```

Output:

```
{
  "data": [
    {
      "id": "abc-123-def-456",
      "path": ["MySpace"],
      "type": "CONTAINER",
      "containerType": "SPACE",
      "createdAt": "2024-01-01T00:00:00Z"
    }
```

```
  ]
}
```

YAML

```
dremio --output yaml catalog list
```

Output:

```
data:
  - id: abc-123-def-456
    path:
      - MySpace
    type: CONTAINER
    containerType: SPACE
    createdAt: '2024-01-01T00:00:00Z'
```

Path Formats

Cloud

```
source.namespace.object
```

Examples:

˜evangelism-2026.testing.my_table`

˜my-s3-source.data.customers`

Software

```
space.object
catalog.namespace.object
```

Examples:

˜Analytics.sales_data`

˜dremio-catalog.alexmerced.testing`

˜@user@company.com.my_view`

Common Use Cases

1. Inventory Management

```
# Export full catalog inventory
dremio --output json catalog list > catalog_inventory.json

# Count items by type
dremio --output json catalog list | jq '[.data[] | .containerType] | group_by(.) |
map({type: .[0], count: length})'
```

2. Finding Specific Items

```
# Find all spaces
dremio --output json catalog list | jq '.data[] | select(.containerType == "SPACE")'

# Find all sources
dremio --output json catalog list | jq '.data[] | select(.containerType == "SOURCE")'

# Find all views
dremio --output json catalog list | jq '.data[] | select(.type == "VIRTUAL_DATASET")'
```

3. Validation

```
# Verify item exists
dremio catalog get-by-path "MySpace.MyTable" && echo "Exists" || echo "Not found"

# Check if path is accessible
dremio catalog get-by-path "Sales.customers" --include permissions
```

4. Migration Planning

```
# List all items in source environment
dremio --profile source catalog list --include datasetCount > source_catalog.json

# List all items in target environment
dremio --profile target catalog list --include datasetCount > target_catalog.json

# Compare and plan migration
diff source_catalog.json target_catalog.json
```

Tips

Use JSON output for scripting:

```
dremio --output json catalog list | jq '.data[] | .path'
```

Filter results with grep:

```
dremio catalog list | grep "Analytics"
```

Save catalog snapshots:

```
dremio --output json catalog list > catalog_$(date +%Y%m%d).json
```

Check permissions before operations:

```
dremio catalog get-by-path "MySpace.MyTable" --include permissions
```

Error Handling

Item Not Found

```
$ dremio catalog get-by-path "NonExistent.Table"
Error: Resource not found
```

Solution: Verify the path exists:

```
dremio catalog list | grep "NonExistent"
```

Permission Denied

```
$ dremio catalog get abc-123
Error: Access forbidden
```

Solution: Check your profile has appropriate permissions.

Invalid Path Format

```
$ dremio catalog get-by-path "Invalid Path With Spaces"
Error: Invalid path format
```

Solution: Use proper path separators:

```
dremio catalog get-by-path "Space.Folder.Object"
```

Source: grants.md

Grant and Privilege Management

This guide covers grant and privilege management for controlling access to catalog objects in Dremio.

Overview

Grants control who can access catalog objects and what operations they can perform. Grants can be assigned to:

Users: Individual user accounts

Roles: Groups of users with shared permissions

Privilege Types

Common privileges include:

`SELECT` - Read data from datasets

`VIEW_REFLECTION` - View reflection metadata

`ALTER` - Modify object metadata

`MODIFY` - Modify object data/structure

`MANAGE_GRANTS` - Manage permissions on the object

`READ_METADATA` - Read object metadata

`CREATE_TABLE` - Create tables in the object

`DROP` - Delete the object

Commands

List Grants

List all grants for a catalog object.

```
dremio grant list <CATALOG_ID>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Examples:

```
# List grants for a space
dremio grant list abc-123-def-456

# List in JSON format
dremio --output json grant list abc-123-def-456
```

Add Grant

Add a grant to a catalog object.

```
dremio grant add <CATALOG_ID> --grantee-type <TYPE> --grantee-id <ID> --privileges <PRIVS>
```

Arguments:

˜CATALOG_ID` - The catalog object ID (UUID)

Options:

˜grantee-type` - Grantee type: `USER` or `ROLE` (required)

˜grantee-id` - User or role ID (required)

˜privileges` - Comma-separated privileges (required)

Examples:

```
# Grant SELECT to a user
dremio grant add abc-123 --grantee-type USER --grantee-id user-456 --privileges SELECT

# Grant multiple privileges to a role
dremio grant add abc-123 --grantee-type ROLE --grantee-id role-789 --privileges SELECT,ALTER,MODIFY

# Grant read-only access
dremio grant add abc-123 --grantee-type USER --grantee-id user-456 --privileges SELECT,VIEW_REFLECTION,READ_METADATA
```

Remove Grant

Remove a grant from a catalog object.

```
dremio grant remove <CATALOG_ID> --grantee-type <TYPE> --grantee-id <ID>
```

Arguments:

˜CATALOG_ID` - The catalog object ID (UUID)

Options:

˜grantee-type` - Grantee type: `USER` or `ROLE` (required)

˜grantee-id` - User or role ID (required)

Examples:

```
# Remove grant from user
dremio grant remove abc-123 --grantee-type USER --grantee-id user-456
```

```
# Remove grant from role (with confirmation)
dremio grant remove abc-123 --grantee-type ROLE --grantee-id role-789

# Remove without confirmation
dremio grant remove abc-123 --grantee-type USER --grantee-id user-456 --yes
```

Set Grants

Set all grants for a catalog object (replaces existing).

```
dremio grant set <CATALOG_ID> --from-file <FILE>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Options:

`from-file` - JSON file with complete grants definition (required)

Examples:

```
# Set grants from file
dremio grant set abc-123 --from-file grants.json
```

Grant File Format

Example grants.json

```
{
  "grants": [
    {
      "granteeType": "USER",
      "granteeId": "user-123",
      "privileges": ["SELECT", "VIEW_REFLECTION"]
    },
    {
      "granteeType": "ROLE",
      "granteeId": "role-456",
      "privileges": ["SELECT", "ALTER", "MODIFY"]
    },
    {
      "granteeType": "ROLE",
      "granteeId": "admin-role",
      "privileges": ["SELECT", "ALTER", "MODIFY", "MANAGE_GRANTS", "DROP"]
    }
  ]
}
```

Scenarios

Granting Read Access to a Dataset

```
# 1. Get dataset ID
DATASET_ID=$(dremio --output json view get-by-path "Analytics.sales_data" | jq -r '.id')

# 2. Grant SELECT to analyst role
dremio grant add $DATASET_ID --grantee-type ROLE --grantee-id analyst-role --privileges
SELECT,VIEW_REFLECTION

# 3. Verify grant
dremio grant list $DATASET_ID
```

Setting Up Role-Based Access

```
# Create grants file for a space
cat > space_grants.json <<EOF
{
  "grants": [
    {
      "granteeType": "ROLE",
      "granteeId": "analyst",
      "privileges": ["SELECT", "VIEW_REFLECTION", "READ_METADATA"]
    },
    {
      "granteeType": "ROLE",
      "granteeId": "data_engineer",
      "privileges": ["SELECT", "ALTER", "MODIFY", "CREATE_TABLE"]
    },
    {
      "granteeType": "ROLE",
      "granteeId": "admin",
              "privileges":  ["SELECT",  "ALTER",  "MODIFY",  "MANAGE_GRANTS",  "DROP",
"CREATE_TABLE"]
    }
  ]
}
EOF

# Apply grants
SPACE_ID=$(dremio  --output  json  space  list  |  jq  -r  '.[]  |  select(.path[0]  ==
"Analytics") | .id')
dremio grant set $SPACE_ID --from-file space_grants.json
```

Migrating Grants

```
# Export grants from source
SOURCE_ID=$(dremio --profile source --output json view get-by-path "Analytics.summary" |
```

```
jq -r '.id')
dremio --profile source --output json grant list $SOURCE_ID > grants_export.json

# Apply to target
TARGET_ID=$(dremio --profile target --output json view get-by-path "Analytics.summary" |
jq -r '.id')
dremio --profile target grant set $TARGET_ID --from-file grants_export.json
```

Common Workflows

1. Audit Access

```
#!/bin/bash
# audit_access.sh - Audit grants across catalog

# Get all spaces
dremio --output json space list | jq -r '.[].id' | while read space_id; do
  echo "Space: $space_id"
    dremio  --output  json  grant  list  $space_id  |  jq  '.grants[]  |  "\(.granteeType):
\(.granteeId) - \(.privileges | join(", "))"'
  echo ""
done
```

2. Bulk Grant Assignment

```
#!/bin/bash
# grant_to_all_views.sh - Grant access to all views in a space

SPACE="Analytics"
ROLE_ID="analyst-role"
PRIVILEGES="SELECT,VIEW_REFLECTION"

# Get all views in space
dremio --output json view list --space $SPACE | jq -r '.[].id' | while read view_id; do
  echo "Granting to view: $view_id"
   dremio  grant  add  $view_id  --grantee-type  ROLE  --grantee-id  $ROLE_ID  --privileges
$PRIVILEGES
done
```

3. Remove User Access

```
#!/bin/bash
# revoke_user_access.sh - Remove all grants for a user

USER_ID="user-123"

# Find all objects with grants
```

```
dremio --output json catalog list | jq -r '.data[].id' | while read object_id; do
  # Check if user has grants
  if dremio --output json grant list $object_id | jq -e ".grants[] | select(.granteeId
== \"$USER_ID\")" > /dev/null; then
    echo "Removing grant from: $object_id"
    dremio grant remove $object_id --grantee-type USER --grantee-id $USER_ID --yes
  fi
done
```

4. Grant Templates

```
#!/bin/bash
# apply_grant_template.sh - Apply standard grant template

TEMPLATE=$1  # read-only, read-write, or admin
OBJECT_ID=$2

case $TEMPLATE in
  read-only)
    cat > grants.json <<EOF
{
  "grants": [
    {
      "granteeType": "ROLE",
      "granteeId": "viewer",
      "privileges": ["SELECT", "VIEW_REFLECTION", "READ_METADATA"]
    }
  ]
}
EOF
    ;;
  read-write)
    cat > grants.json <<EOF
{
  "grants": [
    {
      "granteeType": "ROLE",
      "granteeId": "editor",
      "privileges": ["SELECT", "ALTER", "MODIFY", "CREATE_TABLE"]
    }
  ]
}
EOF
    ;;
  admin)
    cat > grants.json <<EOF
{
  "grants": [
    {
      "granteeType": "ROLE",
      "granteeId": "admin",
              "privileges":  ["SELECT",  "ALTER",  "MODIFY",  "MANAGE_GRANTS",  "DROP",
```

```
"CREATE_TABLE"]
    }
  ]
}
EOF
    ;;
esac

dremio grant set $OBJECT_ID --from-file grants.json
rm grants.json
```

Tips

Use roles over users: Assign grants to roles for easier management

```
dremio grant add $ID --grantee-type ROLE --grantee-id analyst --privileges SELECT
```

Principle of least privilege: Grant minimum necessary permissions

```
# Good: specific privileges
dremio grant add $ID --grantee-type USER --grantee-id user-123 --privileges SELECT

# Avoid: excessive privileges
dremio grant add $ID --grantee-type USER --grantee-id user-123 --privileges
SELECT,ALTER,MODIFY,DROP
```

Document grant decisions: Add wiki documentation

```
dremio wiki set $ID --text "# Access Control\n\nAnalyst role has read-only access"
```

Regular audits: Review grants periodically

```
# Export current grants for review
dremio --output json grant list $ID > grants_$(date +%Y%m%d).json
```

Error Handling

Insufficient Permissions

```
$ dremio grant add abc-123 --grantee-type USER --grantee-id user-456 --privileges SELECT
Error: Insufficient permissions to manage grants
```

Solution: Ensure you have `MANAGE_GRANTS` privilege on the object.

Invalid Privilege

```
$ dremio grant add abc-123 --grantee-type USER --grantee-id user-456 --privileges INVALID
Error: Invalid privilege: INVALID
```

Solution: Use valid privilege names (SELECT, ALTER, MODIFY, etc.).

Grantee Not Found

```
$ dremio grant add abc-123 --grantee-type USER --grantee-id invalid-user --privileges SELECT
Error: User not found: invalid-user
```

Solution: Verify the user/role ID exists.

Platform Differences

Software

- Full grant management support
- User and role-based grants
- All privilege types available

Cloud

- Grant management available
- May have different privilege types
- Project-scoped permissions

Best Practices

Use role-based access control: Assign grants to roles, not individual users

Least privilege principle: Grant minimum necessary permissions

Regular audits: Review and update grants periodically

Document access policies: Use wiki to document why grants exist

- **Test before production**: Verify grants in dev/staging first
- **Backup grants**: Export grant configurations before changes
- **Automate common patterns**: Use scripts for standard grant templates

Monitor access: Track who has access to sensitive data

Privilege Reference

Data Access

`SELECT` - Query data

`VIEW_REFLECTION` - View reflection metadata

`READ_METADATA` - Read object metadata

Data Modification

`ALTER` - Modify metadata

`MODIFY` - Modify data/structure

`CREATE_TABLE` - Create tables

`DROP` - Delete objects

Administration

`MANAGE_GRANTS` - Manage permissions

`OWNERSHIP` - Full control

Advanced Usage

Conditional Grants

```
#!/bin/bash
# conditional_grants.sh - Grant based on conditions

OBJECT_ID=$1
ENVIRONMENT=$2

if [ "$ENVIRONMENT" == "production" ]; then
  # Production: read-only for most users
   dremio grant add $OBJECT_ID --grantee-type ROLE --grantee-id analyst --privileges
SELECT
else
  # Development: read-write
   dremio grant add $OBJECT_ID --grantee-type ROLE --grantee-id analyst --privileges
SELECT,ALTER,MODIFY
fi
```

Grant Inheritance

```
#!/bin/bash
```

```
# inherit_grants.sh - Apply parent grants to children

PARENT_ID=$1

# Get parent grants
dremio --output json grant list $PARENT_ID > parent_grants.json

# Apply to all children
dremio --output json catalog list | jq -r ".data[] | select(.path[0] == \"$PARENT_NAME\") | .id" | while read child_id; do
  dremio grant set $child_id --from-file parent_grants.json
done
```

Grant Reporting

```
#!/bin/bash
# grant_report.sh - Generate grant report

echo "# Grant Report - $(date)"
echo ""

dremio --output json catalog list | jq -r '.data[] | select(.containerType == "SPACE") | .id' | while read space_id; do
  SPACE_NAME=$(dremio --output json catalog get $space_id | jq -r '.path[0]')
  echo "## Space: $SPACE_NAME"
  echo ""

   dremio --output json grant list $space_id | jq -r '.grants[] | "- \(.granteeType): \(.granteeId) - \(.privileges | join(", "))"'
  echo ""
done
```

Summary

List: View all grants on an object

Add: Grant privileges to users/roles

Remove: Revoke access

Set: Replace all grants with new configuration

Use roles: Simplify management

Audit regularly: Maintain security

Document: Explain access decisions

Source: installation.md

Installation Guide

Requirements

- Python 3.8 or higher
- pip or pipx

Installation Methods

Using pip (Recommended for users)

```
pip install dremio-cli
```

Using pipx (Isolated environment)

```
pipx install dremio-cli
```

From Source (For developers)

```
git clone https://github.com/developer-advocacy-dremio/dremio-python-cli
cd dremio-cli
pip install -e .
```

Development Installation

```
git clone https://github.com/developer-advocacy-dremio/dremio-python-cli
cd dremio-cli
pip install -e ".[dev]"
```

Verify Installation

```
dremio --version
```

Next Steps

See the Quick Start Guide to configure your first profile and start using the CLI.

Source: jobs.md

Job Management

This guide covers job management operations including listing, monitoring, and managing Dremio query jobs.

Commands

List Jobs

List recent jobs.

```
dremio job list [OPTIONS]
```

Options:

˜max-results INTEGER` - Maximum number of results to return

˜filter TEXT` - Filter expression (e.g., `state=COMPLETED`)

˜sort TEXT` - Sort field (prefix with `-` for descending, e.g., `-submittedAt`)

Examples:

```
# List recent jobs
dremio job list

# List last 50 jobs
dremio job list --max-results 50

# List only completed jobs
dremio job list --filter "state=COMPLETED"

# List jobs sorted by submission time (newest first)
dremio job list --sort "-submittedAt"

# Combine filters and sorting
dremio job list --max-results 20 --filter "state=RUNNING" --sort "-submittedAt"
```

Get Job Details

Retrieve detailed information about a specific job.

```
dremio job get <JOB_ID>
```

Arguments:

˜JOB_ID` - The job ID (UUID)

Examples:

```
# Get job details
dremio job get 16b2c9cd-a920-952b-b162-2280c9059d00

# Get job details in JSON
dremio --output json job get 16b2c9cd-a920-952b-b162-2280c9059d00

# Get job details with verbose output
dremio --verbose job get 16b2c9cd-a920-952b-b162-2280c9059d00
```

Get Job Results

Retrieve the results of a completed job.

```
dremio job results <JOB_ID> [OPTIONS]
```

Arguments:

˜JOB_ID` - The job ID (UUID)

Options:

˜limit INTEGER` - Maximum number of rows to return

˜offset INTEGER` - Offset for pagination

Examples:

```
# Get job results
dremio job results 16b2c9cd-a920-952b-b162-2280c9059d00

# Get first 100 rows
dremio job results 16b2c9cd-a920-952b-b162-2280c9059d00 --limit 100

# Get next 100 rows (pagination)
dremio job results 16b2c9cd-a920-952b-b162-2280c9059d00 --limit 100 --offset 100

# Export results to JSON
dremio --output json job results 16b2c9cd-a920-952b-b162-2280c9059d00 > results.json
```

Cancel Job

Cancel a running job.

```
dremio job cancel <JOB_ID>
```

Arguments:

˜JOB_ID` - The job ID (UUID)

Examples:

```
# Cancel a running job
dremio job cancel 16b2c9cd-a920-952b-b162-2280c9059d00

# Cancel without confirmation prompt
dremio job cancel 16b2c9cd-a920-952b-b162-2280c9059d00 --yes
```

Get Job Profile

Download job profile for performance analysis.

```
dremio job profile <JOB_ID> [OPTIONS]
```

Arguments:

˜JOB_ID` - The job ID (UUID)

Options:

˜download PATH` - Download profile to file

Examples:

```
# View job profile
dremio job profile 16b2c9cd-a920-952b-b162-2280c9059d00

# Download profile to file
dremio job profile 16b2c9cd-a920-952b-b162-2280c9059d00 --download profile.zip
```

Get Job Reflections

Get reflection information for a job.

```
dremio job reflections <JOB_ID>
```

Arguments:

˜JOB_ID` - The job ID (UUID)

Examples:

```
# Get reflection info
dremio job reflections 16b2c9cd-a920-952b-b162-2280c9059d00

# Get in JSON format
dremio --output json job reflections 16b2c9cd-a920-952b-b162-2280c9059d00
```

Scenarios

Monitoring Query Execution

```
# 1. Execute a query
dremio sql execute "SELECT * FROM large_table LIMIT 1000"
# Output: Job ID: abc-123-def-456

# 2. Check job status
dremio job get abc-123-def-456

# 3. Wait for completion, then get results
dremio job results abc-123-def-456
```

Debugging Slow Queries

```
# 1. List recent jobs
dremio job list --max-results 10

# 2. Get details of slow job
dremio job get slow-job-id

# 3. Download profile for analysis
dremio job profile slow-job-id --download slow_query_profile.zip

# 4. Check if reflections were used
dremio job reflections slow-job-id
```

Pagination Through Large Results

```
# Get results in batches of 1000
for i in {0..9}; do
  offset=$((i * 1000))
  dremio job results abc-123 --limit 1000 --offset $offset > results_part_$i.json
done
```

Monitoring Running Jobs

```
# List all running jobs
dremio job list --filter "state=RUNNING"

# Check specific running job
dremio job get running-job-id

# Cancel if needed
dremio job cancel running-job-id
```

Job History Analysis

```
# Export last 100 jobs
dremio --output json job list --max-results 100 > job_history.json

# Analyze with jq
cat job_history.json | jq '.jobs[] | {id: .id, state: .jobState, duration: .duration}'

# Find failed jobs
cat job_history.json | jq '.jobs[] | select(.jobState == "FAILED")'
```

Job States

Jobs progress through these states:

- **PLANNING** - Query is being planned
- **RUNNING** - Query is executing
- **COMPLETED** - Query finished successfully
- **FAILED** - Query failed
- **CANCELED** - Query was canceled

Common Workflows

1. Execute and Monitor

```
# Execute query
RESULT=$(dremio sql execute "SELECT COUNT(*) FROM customers")
JOB_ID=$(echo $RESULT | grep -oP 'Job ID: \K[a-f0-9-]+')

# Monitor until complete
while true; do
  STATE=$(dremio --output json job get $JOB_ID | jq -r '.jobState')
  echo "Job state: $STATE"
  [[ "$STATE" == "COMPLETED" ]] && break
  sleep 2
done

# Get results
dremio job results $JOB_ID
```

2. Batch Job Management

```
# Get all running jobs
RUNNING_JOBS=$(dremio --output json job list --filter "state=RUNNING" | jq -r
'.jobs[].id')

# Cancel all running jobs
```

```
for job_id in $RUNNING_JOBS; do
  dremio job cancel $job_id --yes
done
```

3. Performance Analysis

```
# Get job details
dremio --output json job get $JOB_ID > job_details.json

# Extract performance metrics
cat job_details.json | jq '{
  duration: .duration,
  rowCount: .rowCount,
  dataProcessed: .dataProcessed,
  reflectionsUsed: .reflectionsUsed
}'

# Download profile for deep analysis
dremio job profile $JOB_ID --download profile_$JOB_ID.zip
```

4. Result Export

```
# Export results to different formats
dremio --output json job results $JOB_ID > results.json
dremio --output yaml job results $JOB_ID > results.yaml
dremio --output table job results $JOB_ID > results.txt

# Convert JSON to CSV
dremio --output json job results $JOB_ID | jq -r '.rows[] | @csv' > results.csv
```

Tips

- **Save job IDs**: Store job IDs for later reference

```
echo "abc-123-def-456" > last_job_id.txt
JOB_ID=$(cat last_job_id.txt)
```

- **Use filters effectively**: Narrow down job lists

```
dremio job list --filter "state=FAILED" --max-results 10
```

- **Automate monitoring**: Create scripts to watch jobs

```
watch -n 5 'dremio job list --max-results 5'
```

-

Export for analysis: Use JSON output for processing

```
dremio --output json job list > jobs.json
```

Error Handling

Job Not Found

```
$ dremio job get invalid-job-id
Error: Resource not found
```

Solution: Verify the job ID is correct.

Results Not Available

```
$ dremio job results abc-123
Error: Cannot fetch results for job in PLANNING state
```

Solution: Wait for job to complete:

```
dremio job get abc-123  # Check state
```

Permission Denied

```
$ dremio job get abc-123
Error: Access forbidden
```

Solution: Ensure you have permission to view the job.

Platform Differences

Cloud

- Job listing may have rate limits
- Some job profile features may differ

Software

- Full job history available
- Complete profile download support
- Reflection information available

Best Practices

Monitor long-running queries: Check job status periodically

Cancel unnecessary jobs: Free up resources

Download profiles for analysis: Investigate performance issues

Use pagination for large results: Avoid memory issues

Filter job lists: Focus on relevant jobs

Export job history: Track query patterns over time

Source: profiles.md

Profile Management Guide

This guide covers how to create and manage Dremio CLI profiles using both YAML configuration files and environment variables.

Overview

Profiles store connection information for Dremio instances. The CLI supports two methods:

YAML Configuration - Stored in `~/.dremio/profiles.yaml`

Environment Variables - Loaded from `.env` file or shell environment

Environment variables take precedence over YAML profiles.

YAML Configuration

Location

Profiles are stored in: `~/.dremio/profiles.yaml`

Creating Profiles via CLI

```
# Create a Dremio Cloud profile
dremio profile create \
  --name cloud-prod \
  --type cloud \
  --base-url https://api.dremio.cloud/v0 \
  --auth-type pat \
  --token YOUR_PERSONAL_ACCESS_TOKEN \
  --project-id YOUR_PROJECT_ID

# Create a Dremio Software profile
```

```
dremio profile create \
  --name software-dev \
  --type software \
  --base-url https://dremio.company.com/api/v3 \
  --auth-type pat \
  --token YOUR_PERSONAL_ACCESS_TOKEN
```

Manual YAML Configuration

Edit `~/.dremio/profiles.yaml`:

```
profiles:
  cloud-prod:
    type: cloud
    base_url: https://api.dremio.cloud/v0
    project_id: 788baab4-3c3b-42da-9f1d-5cc6dc03147d
    auth:
      type: pat
      token: YOUR_ENCRYPTED_TOKEN
    testing_folder: testing

  software-dev:
    type: software
    base_url: https://dremio.company.com/api/v3
    auth:
      type: pat
      token: YOUR_ENCRYPTED_TOKEN
    testing_folder: '"dremio-catalog".alexmerced.testing'

  software-local:
    type: software
    base_url: http://localhost:9047/api/v3
    auth:
      type: username_password
      username: admin
      password: YOUR_ENCRYPTED_PASSWORD

default_profile: cloud-prod
```

Profile Fields

Field	Required	Description
`type`	Yes	`cloud` or `software`
`base_url`	Yes	API endpoint URL
`project_id`	Cloud only	Project UUID
`auth.type`	Yes	`pat`, `oauth`, or `username_password`
`auth.token`	For PAT	Personal Access Token
`auth.username`	For user/pass	Username
`auth.password`	For user/pass	Password

| `testing_folder` | No | Default folder for testing |

Environment Variable Configuration

Pattern

Environment variables follow the pattern:

```
DREMIO_{PROFILE}_{KEY}=value
```

Example .env File

Create a `.env` file in your project directory or home directory:

```
# Cloud Profile
DREMIO_CLOUD_TYPE=cloud
DREMIO_CLOUD_BASE_URL=https://api.dremio.cloud/v0
DREMIO_CLOUD_PROJECTID=788baab4-3c3b-42da-9f1d-5cc6dc03147d
DREMIO_CLOUD_TOKEN=s3JcLOqFTR6qnurWp09epkXfy0+06N9i5oSwG0KbRthqmgiL1DvMgd2+LSNgUA==
DREMIO_CLOUD_TESTING_FOLDER=testing

# Software Profile
DREMIO_SOFTWARE_TYPE=software
DREMIO_SOFTWARE_BASE_URL=https://v26.dremio.org/api/v3
DREMIO_SOFTWARE_TOKEN=Q/ToosxORAuvy2zBLL+Q9O9JCnJL/8KKrsiC1Np3UL8yxQ3IyzGzgoBo2LwzvQ==
DREMIO_SOFTWARE_TESTING_FOLDER='"dremio-catalog".alexmerced.testing'

# Local Development Profile
DREMIO_LOCAL_TYPE=software
DREMIO_LOCAL_BASE_URL=http://localhost:9047/api/v3
DREMIO_LOCAL_USERNAME=admin
DREMIO_LOCAL_PASSWORD=password123
```

Supported Environment Variables

Variable Pattern	Description	Example
`DREMIO_{PROFILE}_TYPE`	Profile type	`cloud` or `software`
`DREMIO_{PROFILE}_BASE_URL`	API endpoint	`https://api.dremio.cloud/v0`
`DREMIO_{PROFILE}_PROJECTID`	Project ID (Cloud)	`788baab4-...`
`DREMIO_{PROFILE}_TOKEN`	Personal Access Token	`s3JcLOqFTR...`
`DREMIO_{PROFILE}_USERNAME`	Username (user/pass auth)	`admin`
`DREMIO_{PROFILE}_PASSWORD`	Password (user/pass auth)	`password123`
`DREMIO_{PROFILE}_TESTING_FOLDER`	Default test folder	`testing`

Loading Environment Variables

The CLI automatically loads `.env` files from:

Current working directory

Home directory (`~/.env`)

You can also set environment variables in your shell:

```
export DREMIO_PROD_TYPE=cloud
export DREMIO_PROD_BASE_URL=https://api.dremio.cloud/v0
export DREMIO_PROD_TOKEN=your_token_here
```

Profile Management Commands

List Profiles

```
# List all profiles (YAML + environment variables)
dremio profile list

# Output formats
dremio --output json profile list
dremio --output yaml profile list
```

View Current Profile

```
# Show the default profile
dremio profile current
```

Set Default Profile

```
# Set default profile in YAML
dremio profile set-default cloud-prod
```

Delete Profile

```
# Delete a YAML profile
dremio profile delete software-dev
```

Note: Environment variable profiles cannot be deleted via CLI.

Using Profiles

Specify Profile for Commands

```
# Use specific profile
```

```
dremio --profile cloud-prod catalog list
dremio --profile software-dev sql execute "SELECT 1"

# Use default profile (no --profile flag)
dremio catalog list
```

Profile Priority

When multiple profiles exist with the same name:

Environment Variables (highest priority)

YAML Configuration

Example:

```
# If both exist, environment variable wins
DREMIO_CLOUD_TOKEN=env_token  # This is used
# vs
profiles.yaml: cloud.auth.token: yaml_token  # This is ignored
```

Security Best Practices

1. Never Commit Credentials

Add to `.gitignore`:

```
.env
.env.*
!.env.example
.dremio/
```

2. Use Environment Variables for CI/CD

```
# GitHub Actions example
env:
  DREMIO_PROD_TYPE: cloud
  DREMIO_PROD_BASE_URL: ${{ secrets.DREMIO_BASE_URL }}
  DREMIO_PROD_TOKEN: ${{ secrets.DREMIO_TOKEN }}
  DREMIO_PROD_PROJECTID: ${{ secrets.DREMIO_PROJECT_ID }}
```

3. Token Encryption

YAML profiles automatically encrypt tokens. Environment variables are stored as-is.

4. Rotate Tokens Regularly

```
# Update token in YAML
dremio profile create --name cloud-prod --token NEW_TOKEN

# Update environment variable
export DREMIO_CLOUD_TOKEN=NEW_TOKEN
```

Example Workflows

Development Workflow

```
# .env file for local development
DREMIO_DEV_TYPE=software
DREMIO_DEV_BASE_URL=http://localhost:9047/api/v3
DREMIO_DEV_USERNAME=admin
DREMIO_DEV_PASSWORD=password123

# Use in commands
dremio --profile dev catalog list
dremio --profile dev sql execute "SELECT * FROM my_table LIMIT 10"
```

Production Workflow

```
# profiles.yaml for production
profiles:
  prod:
    type: cloud
    base_url: https://api.dremio.cloud/v0
    project_id: YOUR_PROJECT_ID
    auth:
      type: pat
      token: ENCRYPTED_TOKEN

# Use in commands
dremio --profile prod job list
dremio --profile prod view create --path '["Analytics", "Summary"]' --sql "..."
```

Multi-Environment Setup

```
# .env file with multiple environments
DREMIO_DEV_TYPE=software
DREMIO_DEV_BASE_URL=http://localhost:9047/api/v3
DREMIO_DEV_TOKEN=dev_token

DREMIO_STAGING_TYPE=cloud
DREMIO_STAGING_BASE_URL=https://api.dremio.cloud/v0
```

```
DREMIO_STAGING_PROJECTID=staging_project_id
DREMIO_STAGING_TOKEN=staging_token

DREMIO_PROD_TYPE=cloud
DREMIO_PROD_BASE_URL=https://api.dremio.cloud/v0
DREMIO_PROD_PROJECTID=prod_project_id
DREMIO_PROD_TOKEN=prod_token

# Switch between environments
dremio --profile dev catalog list
dremio --profile staging catalog list
dremio --profile prod catalog list
```

Troubleshooting

Profile Not Found

```
# List all available profiles
dremio profile list

# Check environment variables
env | grep DREMIO_
```

Authentication Errors

```
# Verify token is valid
dremio --profile cloud-prod --verbose catalog list

# Check base URL is correct
dremio profile list
```

Environment Variables Not Loading

```
# Verify .env file location
ls -la .env

# Check .env file format (no spaces around =)
cat .env

# Manually load .env
export $(cat .env | xargs)
```

Advanced Configuration

Custom .env Location

```
# Set custom .env file path
export DREMIO_ENV_FILE=/path/to/custom/.env
```

Profile Inheritance

Environment variables can override specific YAML fields:

```
# profiles.yaml
profiles:
  cloud:
    type: cloud
    base_url: https://api.dremio.cloud/v0
    project_id: default_project
```

```
# Override project_id via environment
export DREMIO_CLOUD_PROJECTID=different_project

# Now uses different_project instead of default_project
dremio --profile cloud catalog list
```

Summary

YAML: Best for persistent, encrypted profiles

Environment Variables: Best for CI/CD, temporary configs, and overrides

Priority: Environment variables > YAML

Security: Never commit credentials, use `.gitignore`

Flexibility: Mix and match YAML and environment variables as needed

Source: quickstart.md

Quick Start Guide

1. Create Your First Profile

For Dremio Cloud

```
dremio profile create production \
  --type cloud \
  --base-url https://api.dremio.cloud/v0 \
  --project-id your-project-id \
```

```
  --auth-type pat \
  --token your-personal-access-token
```

For Dremio Software

```
dremio profile create local \
  --type software \
  --base-url http://localhost:9047/api/v3 \
  --auth-type username_password \
  --username dremio \
  --password dremio123
```

2. Verify Your Profile

```
# List all profiles
dremio profile list

# Show current profile
dremio profile current
```

3. Run Your First Commands

List Catalog

```
dremio catalog list
```

Execute SQL

```
dremio sql execute "SELECT * FROM MySource.MyTable LIMIT 10"
```

List Sources

```
dremio source list
```

4. Try Interactive Mode

```
dremio repl
```

In REPL mode, you can run commands without the `dremio` prefix:

```
dremio> catalog list
dremio> sql execute "SELECT COUNT(*) FROM MyTable"
dremio> exit
```

5. Explore More Commands

```
# Get help for any command
dremio --help
dremio catalog --help
dremio source --help

# Use different output formats
dremio catalog list --output json
dremio catalog list --output yaml
```

Next Steps

- Browse the Command Reference for detailed documentation
- Check out Examples for common use cases
- Learn about Profile Management

Source: roles.md

Role Management

This guide covers role management operations for administering roles and role memberships in Dremio.

Overview

Role Management allows administrators to create roles, assign users to roles, and manage role-based access control. This is primarily available in Dremio Software.

Commands

List Roles

```
dremio role list
```

Get Role

```
dremio role get <ROLE_ID>
```

Create Role

```
dremio role create --name "Analyst"
dremio role create --from-file role.json
```

Update Role

```
dremio role update <ROLE_ID> --from-file updated_role.json
```

Delete Role

```
dremio role delete <ROLE_ID>
```

Add Member

```
dremio role add-member <ROLE_ID> --user <USER_ID>
```

Remove Member

```
dremio role remove-member <ROLE_ID> --user <USER_ID>
```

Examples

```
# List all roles
dremio role list

# Create role
dremio role create --name "Data Analyst"

# Add user to role
dremio role add-member role-123 --user user-456

# Remove user from role
dremio role remove-member role-123 --user user-456

# Delete role
dremio role delete role-123
```

Role File Format

```
{
  "name": "Data Analyst",
```

```
  "description": "Analysts with read access to datasets"
}
```

Workflows

Role-Based Access Control

```
# 1. Create roles
dremio role create --name "Analyst"
dremio role create --name "Engineer"

# 2. Add users to roles
dremio role add-member analyst-role-id --user user-1
dremio role add-member engineer-role-id --user user-2

# 3. Grant permissions to roles
dremio grant add dataset-id --grantee-type ROLE --grantee-id analyst-role-id
--privileges SELECT
dremio grant add dataset-id --grantee-type ROLE --grantee-id engineer-role-id
--privileges SELECT,ALTER,MODIFY
```

Notes

- Role management requires administrative privileges
- Primarily available in Dremio Software
- Cloud has different role management (via cloud console)
- Use roles with grant management for access control

Source: sources.md

Source Management

This guide covers source management operations for connecting to and managing external data sources in Dremio.

Overview

Sources are connections to external data systems like databases, object storage, and data lakes. Dremio supports many source types including:

- **Databases**: PostgreSQL, MySQL, Oracle, SQL Server, MongoDB
- **Object Storage**: S3, Azure Blob Storage, Google Cloud Storage
-

Data Lakes: Hive, Iceberg, Delta Lake

Cloud Warehouses: Snowflake, Redshift, BigQuery

Commands

List Sources

List all configured sources.

```
dremio source list
```

Examples:

```
# List all sources
dremio source list

# JSON output
dremio --output json source list

# YAML output
dremio --output yaml source list
```

Get Source

Retrieve source details by ID.

```
dremio source get <SOURCE_ID>
```

Arguments:

`SOURCE_ID` - The source ID (UUID)

Examples:

```
# Get source details
dremio source get 791ee75c-956e-40fe-b2cc-0922a0f9b0b4

# Get in JSON format
dremio --output json source get 791ee75c-956e-40fe-b2cc-0922a0f9b0b4
```

Create Source

Create a new data source.

```
dremio source create --name <NAME> --type <TYPE> --config-file <FILE>
```

Options:

`name TEXT` - Source name (required)

`type TEXT` - Source type (required, e.g., POSTGRES, S3, MONGO)

`config-file PATH` - JSON configuration file (required)

Examples:

```
# Create PostgreSQL source
dremio source create --name MyPostgres --type POSTGRES --config-file postgres.json

# Create S3 source
dremio source create --name MyS3 --type S3 --config-file s3.json

# Create MongoDB source
dremio source create --name MyMongo --type MONGO --config-file mongo.json
```

Update Source

Update an existing source configuration.

```
dremio source update <SOURCE_ID> --config-file <FILE>
```

Arguments:

`SOURCE_ID` - The source ID (UUID)

Options:

`config-file PATH` - Updated JSON configuration file (required)

Examples:

```
# Update source configuration
dremio source update abc-123 --config-file updated_postgres.json
```

Refresh Source

Refresh source metadata to discover new tables/files.

```
dremio source refresh <SOURCE_ID>
```

Arguments:

`SOURCE_ID` - The source ID (UUID)

Examples:

```
# Refresh source metadata
```

```
dremio source refresh 791ee75c-956e-40fe-b2cc-0922a0f9b0b4
```

Delete Source

Delete a source.

```
dremio source delete <SOURCE_ID>
```

Arguments:

`SOURCE_ID` - The source ID (UUID)

Options:

`tag TEXT` - Version tag for optimistic concurrency control

Examples:

```
# Delete source (with confirmation)
dremio source delete abc-123

# Delete without confirmation
dremio source delete abc-123 --yes
```

Test Connection

Test a source configuration before creating.

```
dremio source test-connection --config-file <FILE>
```

Options:

`config-file PATH` - JSON configuration file to test (required)

Examples:

```
# Test PostgreSQL connection
dremio source test-connection --config-file postgres.json

# Test S3 connection
dremio source test-connection --config-file s3.json
```

Configuration Examples

PostgreSQL

```
{
```

```
  "hostname": "postgres.company.com",
  "port": 5432,
  "databaseName": "analytics",
  "username": "dremio_user",
  "password": "secure_password",
  "authenticationType": "MASTER"
}
```

S3

```
{
  "credentialType": "ACCESS_KEY",
  "accessKey": "AKIAIOSFODNN7EXAMPLE",
  "accessSecret": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
  "secure": true,
  "externalBucketList": ["my-bucket"],
  "enableAsync": true
}
```

MongoDB

```
{
  "host": "mongodb.company.com",
  "port": 27017,
  "authenticationType": "MASTER",
  "username": "dremio_user",
  "password": "secure_password",
  "authDatabase": "admin",
  "useSsl": true
}
```

Azure Blob Storage

```
{
  "accountKind": "STORAGE_V2",
  "accountName": "mystorageaccount",
  "accessKey": "account_access_key",
  "secure": true,
  "containers": ["data-container"]
}
```

Scenarios

Creating a New PostgreSQL Source

```
# 1. Create configuration file
cat > postgres.json <<EOF
{
  "hostname": "postgres.company.com",
  "port": 5432,
  "databaseName": "analytics",
  "username": "dremio_user",
  "password": "secure_password",
  "authenticationType": "MASTER"
}
EOF

# 2. Test connection
dremio source test-connection --config-file postgres.json

# 3. Create source
dremio source create --name Analytics_DB --type POSTGRES --config-file postgres.json

# 4. Get source ID
SOURCE_ID=$(dremio --output json source list | jq -r '.[] | select(.path[0] ==
"Analytics_DB") | .id')

# 5. Refresh metadata
dremio source refresh $SOURCE_ID
```

Updating Source Credentials

```
# 1. Get current source
dremio --output json source get abc-123 > current_config.json

# 2. Edit configuration
cat current_config.json | jq '.config.password = "new_password"' > updated_config.json

# 3. Update source
dremio source update abc-123 --config-file updated_config.json

# 4. Test connection
dremio source refresh abc-123
```

Migrating Sources

```
# 1. Export source from old environment
dremio --profile old --output json source get abc-123 > source_export.json

# 2. Extract configuration
cat source_export.json | jq '.config' > source_config.json

# 3. Create in new environment
dremio --profile new source create \
  --name $(cat source_export.json | jq -r '.path[0]') \
```

```
  --type $(cat source_export.json | jq -r '.type') \
  --config-file source_config.json
```

Common Workflows

1. Bulk Source Creation

```
#!/bin/bash
# create_sources.sh

# Define sources
declare -A SOURCES=(
  ["Postgres_Prod"]="POSTGRES:postgres_prod.json"
  ["S3_DataLake"]="S3:s3_datalake.json"
  ["Mongo_Events"]="MONGO:mongo_events.json"
)

# Create each source
for name in "${!SOURCES[@]}"; do
  IFS=':' read -r type config <<< "${SOURCES[$name]}"

  echo "Creating source: $name"
  dremio source create --name "$name" --type "$type" --config-file "$config"
done
```

2. Source Health Check

```
#!/bin/bash
# check_sources.sh

# Get all sources
SOURCES=$(dremio --output json source list | jq -r '.[].id')

for source_id in $SOURCES; do
  echo "Checking source: $source_id"

  # Try to refresh
  if dremio source refresh $source_id 2>/dev/null; then
    echo "  ✓ Healthy"
  else
    echo "  ✗ Unhealthy"
  fi
done
```

3. Automated Refresh

```
#!/bin/bash
```

```
# refresh_all_sources.sh

# Refresh all sources nightly
dremio --output json source list | jq -r '.[].id' | while read source_id; do
  echo "Refreshing source: $source_id"
  dremio source refresh $source_id
  sleep 5  # Rate limiting
done
```

4. Source Inventory

```
# Export source inventory
dremio --output json source list | jq '[.[] | {
  name: .path[0],
  type: .type,
  id: .id,
  created: .createdAt
}]' > source_inventory.json

# Generate report
cat source_inventory.json | jq -r '.[] | "\(.name) (\(.type))"'
```

Tips

Test before creating: Always test connections first

```
dremio source test-connection --config-file config.json
```

Store configs securely: Don't commit credentials to git

```
# Add to .gitignore
echo "*.source.json" >> .gitignore
```

Use environment variables: For sensitive data

```
# In config file, use placeholders
cat > postgres.json <<EOF
{
  "hostname": "${POSTGRES_HOST}",
  "username": "${POSTGRES_USER}",
  "password": "${POSTGRES_PASSWORD}"
}
EOF

# Substitute before use
envsubst < postgres.json > postgres_final.json
```

-

Regular refreshes: Keep metadata up-to-date

```
# Cron job for daily refresh
0 2 * * * dremio source refresh abc-123
```

Error Handling

Connection Test Failed

```
$ dremio source test-connection --config-file postgres.json
✗ Connection test failed
  Error: Connection refused
```

Solution: Check hostname, port, and network access:

```
# Test connectivity
telnet postgres.company.com 5432

# Check credentials
psql -h postgres.company.com -U dremio_user -d analytics
```

Source Already Exists

```
$ dremio source create --name MyDB --type POSTGRES --config-file config.json
Error: Source with name 'MyDB' already exists
```

Solution: Use a different name or update existing source:

```
# Get existing source ID
SOURCE_ID=$(dremio --output json source list | jq -r '.[] | select(.path[0] == "MyDB") |
.id')

# Update instead
dremio source update $SOURCE_ID --config-file config.json
```

Invalid Configuration

```
$ dremio source create --name MyS3 --type S3 --config-file s3.json
Error: Invalid configuration: missing required field 'accessKey'
```

Solution: Verify configuration format:

```
# Check required fields for source type
cat s3.json | jq '.'
```

Platform Differences

Software

- Full source management support
- All source types available
- Local file sources supported

Cloud

- Managed sources (some types)
- Cloud-native sources (S3, Azure, GCS)
- Some source types may be restricted

Best Practices

- **Test connections**: Always test before creating
- **Secure credentials**: Use secrets management
- **Regular refreshes**: Keep metadata current
- **Monitor health**: Check source status regularly
- **Version control configs**: Track configuration changes
- **Document sources**: Add wiki documentation
- **Tag sources**: Organize with tags
- **Backup configs**: Export source configurations

Source Types Reference

Databases

- `POSTGRES` - PostgreSQL
- `MYSQL` - MySQL
- `ORACLE` - Oracle Database
- `MSSQL` - Microsoft SQL Server
- `MONGO` - MongoDB

Object Storage

`S3` - Amazon S3

`ADLS` - Azure Data Lake Storage

`GCS` - Google Cloud Storage

Data Lakes

`HIVE` - Apache Hive

`ICEBERG` - Apache Iceberg

`DELTALAKE` - Delta Lake

Cloud Warehouses

`SNOWFLAKE` - Snowflake

`REDSHIFT` - Amazon Redshift

`BIGQUERY` - Google BigQuery

Advanced Usage

Dynamic Configuration

```
#!/bin/bash
# generate_source_config.sh

# Generate config from environment
cat > source.json <<EOF
{
  "hostname": "${DB_HOST}",
  "port": ${DB_PORT},
  "databaseName": "${DB_NAME}",
  "username": "${DB_USER}",
  "password": "${DB_PASSWORD}",
  "authenticationType": "MASTER"
}
EOF

# Create source
dremio source create --name "$SOURCE_NAME" --type POSTGRES --config-file source.json

# Clean up
rm source.json
```

Source Monitoring

```
#!/bin/bash
```

```
# monitor_sources.sh

while true; do
  # Check each source
  dremio --output json source list | jq -r '.[].id' | while read id; do
    # Try to get source
    if ! dremio source get $id >/dev/null 2>&1; then
      echo "Alert: Source $id is unavailable"
      # Send notification
    fi
  done

  sleep 300  # Check every 5 minutes
done
```

Summary

List: View all configured sources

Get: Retrieve source details

Create: Add new data sources

Update: Modify source configuration

Refresh: Update metadata

Delete: Remove sources

Test: Validate configuration before creating

Source: spaces-folders.md

Space and Folder Management

This guide covers space and folder management operations for organizing your Dremio catalog.

Overview

Spaces and **Folders** are containers for organizing your data:

Cloud: Spaces are top-level folders in the project catalog

Software: Spaces are traditional SPACE containers, folders are FOLDER containers

The CLI handles these differences transparently.

Space Commands

Create Space

Create a new space.

```
dremio space create --name <NAME> [OPTIONS]
```

Options:

˜name TEXT` - Space name (required)

˜description TEXT` - Space description

Examples:

```
# Create simple space
dremio space create --name "Analytics"

# Create with description
dremio space create --name "Sales" --description "Sales data and reports"

# Cloud: Creates top-level folder
dremio --profile cloud space create --name "Marketing"

# Software: Creates traditional SPACE
dremio --profile software space create --name "DataScience"
```

List Spaces

List all spaces.

```
dremio space list
```

Examples:

```
# List all spaces
dremio space list

# JSON output
dremio --output json space list

# YAML output
dremio --output yaml space list
```

Get Space

Retrieve space details by ID.

```
dremio space get <SPACE_ID>
```

Arguments:

`SPACE_ID` - The space ID (UUID)

Examples:

```
# Get space details
dremio space get 66c76a3e-0335-463b-8622-1720f8546537

# Get in JSON format
dremio --output json space get 66c76a3e-0335-463b-8622-1720f8546537
```

Delete Space

Delete a space.

```
dremio space delete <SPACE_ID> [OPTIONS]
```

Arguments:

`SPACE_ID` - The space ID (UUID)

Options:

`tag TEXT` - Version tag for optimistic concurrency control

Examples:

```
# Delete space (with confirmation)
dremio space delete 66c76a3e-0335-463b-8622-1720f8546537

# Delete without confirmation
dremio space delete 66c76a3e-0335-463b-8622-1720f8546537 --yes

# Delete with specific tag
dremio space delete 66c76a3e-0335-463b-8622-1720f8546537 --tag "version-tag-123"
```

Folder Commands

Create Folder

Create a new folder.

```
dremio folder create --path <PATH> [OPTIONS]
```

Options:

`path TEXT` - Folder path as JSON array or slash-separated (required)

-

`description TEXT` - Folder description

Examples:

```
# Create folder with slash-separated path
dremio folder create --path "Analytics/Reports"

# Create with JSON array path
dremio folder create --path '["Analytics", "Reports", "2024"]'

# Create with description
dremio folder create --path "Sales/Data" --description "Sales data folder"

# Create nested folders
dremio folder create --path "Analytics/Reports/Monthly"
dremio folder create --path "Analytics/Reports/Quarterly"
```

List Folders

List folders.

```
dremio folder list [OPTIONS]
```

Options:

`parent TEXT` - Parent folder/space ID or path

Examples:

```
# List all folders
dremio folder list

# List folders in specific parent
dremio folder list --parent "Analytics"

# List by parent ID
dremio folder list --parent abc-123-def-456

# JSON output
dremio --output json folder list
```

Get Folder

Retrieve folder details by ID.

```
dremio folder get <FOLDER_ID>
```

Arguments:

`FOLDER_ID` - The folder ID (UUID)

Examples:

```
# Get folder details
dremio folder get 116f8103-159d-4640-b64a-68469bcb21b1

# Get in JSON format
dremio --output json folder get 116f8103-159d-4640-b64a-68469bcb21b1
```

Get Folder by Path

Retrieve folder details by path.

```
dremio folder get-by-path <PATH>
```

Arguments:

`PATH` - The folder path (dot-separated or slash-separated)

Examples:

```
# Get by slash-separated path
dremio folder get-by-path "Analytics/Reports"

# Get by dot-separated path
dremio folder get-by-path "Analytics.Reports.Monthly"

# Get by JSON array path
dremio folder get-by-path '["Analytics", "Reports", "2024"]'
```

Delete Folder

Delete a folder.

```
dremio folder delete <FOLDER_ID> [OPTIONS]
```

Arguments:

`FOLDER_ID` - The folder ID (UUID)

Options:

`tag TEXT` - Version tag for optimistic concurrency control

Examples:

```
# Delete folder (with confirmation)
dremio folder delete 116f8103-159d-4640-b64a-68469bcb21b1

# Delete without confirmation
dremio folder delete 116f8103-159d-4640-b64a-68469bcb21b1 --yes
```

Scenarios

Creating an Organized Catalog

```
# 1. Create top-level spaces
dremio space create --name "Raw" --description "Raw data from sources"
dremio space create --name "Curated" --description "Cleaned and transformed data"
dremio space create --name "Analytics" --description "Business analytics views"

# 2. Create folder structure in Raw
dremio folder create --path "Raw/Customers"
dremio folder create --path "Raw/Orders"
dremio folder create --path "Raw/Products"

# 3. Create folder structure in Curated
dremio folder create --path "Curated/Dimensions"
dremio folder create --path "Curated/Facts"

# 4. Create folder structure in Analytics
dremio folder create --path "Analytics/Sales"
dremio folder create --path "Analytics/Marketing"
dremio folder create --path "Analytics/Finance"
```

Medallion Architecture

```
# Bronze layer (raw data)
dremio space create --name "Bronze" --description "Raw data ingestion"
dremio folder create --path "Bronze/source_system_1"
dremio folder create --path "Bronze/source_system_2"

# Silver layer (cleaned data)
dremio space create --name "Silver" --description "Cleaned and validated data"
dremio folder create --path "Silver/customers"
dremio folder create --path "Silver/orders"
dremio folder create --path "Silver/products"

# Gold layer (business aggregates)
dremio space create --name "Gold" --description "Business-ready datasets"
dremio folder create --path "Gold/customer_360"
dremio folder create --path "Gold/sales_metrics"
dremio folder create --path "Gold/inventory_status"
```

Department-Based Organization

```
# Create department spaces
dremio space create --name "Sales" --description "Sales department data"
dremio space create --name "Marketing" --description "Marketing department data"
dremio space create --name "Finance" --description "Finance department data"
```

```
# Create project folders within departments
dremio folder create --path "Sales/Q1_2024"
dremio folder create --path "Sales/Q2_2024"
dremio folder create --path "Marketing/Campaigns"
dremio folder create --path "Marketing/Analytics"
dremio folder create --path "Finance/Reports"
dremio folder create --path "Finance/Forecasts"
```

Migration and Cleanup

```
# List all spaces
dremio --output json space list > spaces.json

# List all folders
dremio --output json folder list > folders.json

# Find empty folders
cat folders.json | jq '.[] | select(.datasetCount == 0)'

# Delete empty folders
for folder_id in $(cat folders.json | jq -r '.[] | select(.datasetCount == 0) | .id');
do
  dremio folder delete $folder_id --yes
done
```

Common Workflows

1. Create Hierarchical Structure

```
# Create parent space
dremio space create --name "DataWarehouse"

# Create level 1 folders
dremio folder create --path "DataWarehouse/Staging"
dremio folder create --path "DataWarehouse/Production"

# Create level 2 folders
dremio folder create --path "DataWarehouse/Staging/Daily"
dremio folder create --path "DataWarehouse/Staging/Weekly"
dremio folder create --path "DataWarehouse/Production/Current"
dremio folder create --path "DataWarehouse/Production/Archive"

# Create level 3 folders
dremio folder create --path "DataWarehouse/Production/Current/2024"
dremio folder create --path "DataWarehouse/Production/Current/2023"
```

2. Batch Folder Creation

```
# Create folders from list
FOLDERS=(
  "Analytics/Reports/Daily"
  "Analytics/Reports/Weekly"
  "Analytics/Reports/Monthly"
  "Analytics/Dashboards/Executive"
  "Analytics/Dashboards/Operational"
)

for folder in "${FOLDERS[@]}"; do
  dremio folder create --path "$folder"
done
```

3. Folder Inventory

```
# Export folder structure
dremio --output json folder list > folder_inventory.json

# Generate tree view
cat folder_inventory.json | jq -r '.[] | .path | join("/")' | sort

# Count folders by parent
cat folder_inventory.json | jq -r '.[] | .path[0]' | sort | uniq -c
```

4. Space and Folder Cleanup

```
# Get space ID
SPACE_ID=$(dremio --output json space list | jq -r '.[] | select(.path[0] == "OldSpace")
| .id')

# List all folders in space
dremio --output json folder list --parent $SPACE_ID > space_folders.json

# Delete all folders (bottom-up)
cat space_folders.json | jq -r '.[] | .id' | tac | while read folder_id; do
  dremio folder delete $folder_id --yes
done

# Delete space
dremio space delete $SPACE_ID --yes
```

Tips

Plan your structure: Design folder hierarchy before creating

```
Space/
├── Category1/
│   ├── Subcategory1/
```

```
│   └── Subcategory2/
└── Category2/
```

Use consistent naming: Follow naming conventions

```
dremio space create --name "analytics"  # lowercase
dremio folder create --path "analytics/reports"  # lowercase
```

Document structure: Keep a README or diagram

```
dremio --output json folder list | jq -r '.[] | .path | join("/")' > structure.txt
```

Clean up regularly: Remove unused folders

```
dremio folder list | grep "old_"
```

Error Handling

Space Already Exists

```
$ dremio space create --name "Analytics"
Error: Space already exists
```

Solution: Use a different name or delete existing space.

Parent Not Found

```
$ dremio folder create --path "NonExistent/folder"
Error: Parent path does not exist
```

Solution: Create parent first:

```
dremio space create --name "NonExistent"
dremio folder create --path "NonExistent/folder"
```

Cannot Delete Non-Empty

```
$ dremio space delete abc-123
Error: Cannot delete non-empty space
```

Solution: Delete contents first:

```
# Delete all folders in space
```

```
dremio folder list --parent abc-123
# Delete each folder, then delete space
```

Platform Differences

Cloud

- Spaces are top-level folders
- Path: `source.namespace.folder`
- Example: `evangelism-2026.Analytics.Reports`

Software

- Spaces are SPACE containers
- Folders are FOLDER containers
- Path: `space.folder` or `catalog.namespace.folder`
- Example: `Analytics.Reports` or `dremio-catalog.namespace.folder`

Best Practices

- **Organize by purpose**: Group related data together
- **Use descriptive names**: Make structure self-documenting
- **Limit nesting depth**: Keep hierarchy manageable (3-4 levels max)
- **Document structure**: Maintain documentation of organization
- **Regular cleanup**: Remove unused spaces and folders
- **Consistent naming**: Follow naming conventions
- **Plan for growth**: Design scalable structure
- **Use folders for projects**: Separate temporary from permanent data

Source: sql.md

SQL Operations

This guide covers SQL query execution, including file-based queries, context management, async execution, and query analysis.

Commands

Execute SQL Query

Execute a SQL query and return results.

```
dremio sql execute <QUERY> [OPTIONS]
dremio sql execute --file <FILE> [OPTIONS]
```

Arguments:

˜QUERY` - SQL query string (optional if using `file`)

Options:

˜file PATH` - Execute SQL from file

˜context TEXT` - Query context (comma-separated path)

˜async` - Execute asynchronously (return job ID immediately)

˜output-file PATH` - Save results to file

Examples:

```
# Execute simple query
dremio sql execute "SELECT * FROM customers LIMIT 10"

# Execute from file
dremio sql execute --file query.sql

# Execute with context
dremio sql execute "SELECT * FROM table" --context "MySpace"

# Async execution (for long-running queries)
dremio sql execute "SELECT * FROM large_table" --async

# Save results to file
dremio sql execute "SELECT * FROM table" --output-file results.json

# Combine options
dremio sql execute --file complex_query.sql --context "Analytics" --output-file
results.json
```

Explain Query

Generate and display the execution plan for a query.

```
dremio sql explain <QUERY> [OPTIONS]
dremio sql explain --file <FILE> [OPTIONS]
```

Arguments:

˜QUERY` - SQL query string (optional if using `file`)

Options:

`file PATH` - Explain SQL from file

`context TEXT` - Query context

Examples:

```
# Explain simple query
dremio sql explain "SELECT * FROM customers WHERE region = 'US'"

# Explain from file
dremio sql explain --file query.sql

# Explain with context
dremio sql explain "SELECT * FROM table" --context "MySpace"
```

Validate Query

Validate SQL query syntax without executing.

```
dremio sql validate <QUERY> [OPTIONS]
dremio sql validate --file <FILE> [OPTIONS]
```

Arguments:

`QUERY` - SQL query string (optional if using `file`)

Options:

`file PATH` - Validate SQL from file

`context TEXT` - Query context

Examples:

```
# Validate query syntax
dremio sql validate "SELECT * FROM customers"

# Validate from file
dremio sql validate --file query.sql

# Validate with context
dremio sql validate "SELECT * FROM table" --context "MySpace"
```

Scenarios

Interactive Query Development

```
# 1. Start with a simple query
dremio sql execute "SELECT * FROM customers LIMIT 5"
```

```
# 2. Validate more complex query
dremio sql validate "SELECT c.*, o.total FROM customers c JOIN orders o ON c.id =
o.customer_id"

# 3. Explain to check performance
dremio sql explain "SELECT c.*, o.total FROM customers c JOIN orders o ON c.id =
o.customer_id"

# 4. Execute and save results
dremio sql execute "SELECT c.*, o.total FROM customers c JOIN orders o ON c.id =
o.customer_id" --output-file results.json
```

File-Based Query Management

```
# Create query file
cat > monthly_sales.sql <<EOF
SELECT
  DATE_TRUNC('month', order_date) as month,
  SUM(amount) as total_sales,
  COUNT(*) as order_count
FROM orders
WHERE order_date >= '2024-01-01'
GROUP BY 1
ORDER BY 1 DESC
EOF

# Validate the query
dremio sql validate --file monthly_sales.sql

# Execute and save results
dremio sql execute --file monthly_sales.sql --output-file monthly_sales.json

# Explain for optimization
dremio sql explain --file monthly_sales.sql
```

Async Execution for Long Queries

```
# Submit long-running query
dremio sql execute "SELECT * FROM huge_table" --async
# Output: Job ID: abc-123-def-456

# Check job status
dremio job get abc-123-def-456

# Get results when ready
dremio job results abc-123-def-456 --output-file results.json
```

Context-Aware Queries

```
# Set context to avoid fully-qualified names
dremio sql execute "SELECT * FROM customers" --context "Sales"

# Instead of:
dremio sql execute "SELECT * FROM Sales.customers"

# Multi-level context
dremio sql execute "SELECT * FROM table" --context "Analytics,Reports"
```

Batch Query Execution

```
# Execute multiple queries
for query_file in queries/*.sql; do
  echo "Executing $query_file..."
  dremio sql execute --file "$query_file" --output-file "results/$(basename $query_file
.sql).json"
done
```

Common Workflows

1. Query Development Cycle

```
# Step 1: Validate syntax
dremio sql validate "SELECT * FROM customers WHERE region = 'US'"

# Step 2: Check execution plan
dremio sql explain "SELECT * FROM customers WHERE region = 'US'"

# Step 3: Test with small dataset
dremio sql execute "SELECT * FROM customers WHERE region = 'US' LIMIT 10"

# Step 4: Execute full query
dremio sql execute "SELECT * FROM customers WHERE region = 'US'" --output-file
us_customers.json
```

2. Performance Analysis

```
# Get execution plan
dremio sql explain "SELECT c.*, SUM(o.amount) FROM customers c JOIN orders o ON c.id =
o.customer_id GROUP BY c.id" > plan.txt

# Execute and time
time dremio sql execute "SELECT c.*, SUM(o.amount) FROM customers c JOIN orders o ON
c.id = o.customer_id GROUP BY c.id" --async
```

```
# Get job details for analysis
dremio job get <job-id>

# Download profile
dremio job profile <job-id> --download profile.zip
```

3. Data Export

```
# Export to JSON
dremio sql execute "SELECT * FROM customers" --output-file customers.json

# Export to YAML
dremio --output yaml sql execute "SELECT * FROM customers" --output-file customers.yaml

# Convert to CSV using jq
dremio --output json sql execute "SELECT * FROM customers" | jq -r '.rows[] | @csv' >
customers.csv
```

4. Scheduled Queries

```
#!/bin/bash
# daily_report.sh

# Execute daily sales query
dremio sql execute --file daily_sales.sql --output-file "reports/sales_$(date
+%Y%m%d).json"

# Execute customer metrics
dremio sql execute --file customer_metrics.sql --output-file "reports/customers_$(date
+%Y%m%d).json"

# Send notification
echo "Daily reports generated" | mail -s "Dremio Reports" admin@company.com
```

SQL File Format

Basic Query File

```
-- monthly_sales.sql
SELECT
  DATE_TRUNC('month', order_date) as month,
  SUM(amount) as total_sales
FROM orders
GROUP BY 1
ORDER BY 1 DESC
```

Complex Query File

```
-- customer_analysis.sql
WITH customer_orders AS (
  SELECT
    customer_id,
    COUNT(*) as order_count,
    SUM(amount) as total_spent
  FROM orders
  WHERE order_date >= '2024-01-01'
  GROUP BY customer_id
),
customer_segments AS (
  SELECT
    customer_id,
    CASE
      WHEN total_spent > 10000 THEN 'Premium'
      WHEN total_spent > 1000 THEN 'Standard'
      ELSE 'Basic'
    END as segment
  FROM customer_orders
)
SELECT
  c.name,
  c.email,
  co.order_count,
  co.total_spent,
  cs.segment
FROM customers c
JOIN customer_orders co ON c.id = co.customer_id
JOIN customer_segments cs ON c.id = cs.customer_id
ORDER BY co.total_spent DESC
```

Output Formats

Table (Default)

```
dremio sql execute "SELECT * FROM customers LIMIT 5"
```

Output:

```
┌────┬────────────┬────────────────────┬──────────┐
│ ID │ Name       │ Email              │ Region   │
├────┼────────────┼────────────────────┼──────────┤
│ 1  │ John Doe   │ john@email.com     │ US       │
│ 2  │ Jane Doe   │ jane@email.com     │ EU       │
└────┴────────────┴────────────────────┴──────────┘
```

JSON

```
dremio --output json sql execute "SELECT * FROM customers LIMIT 2"
```

Output:

```
{
  "rows": [
    {"id": 1, "name": "John Doe", "email": "john@email.com"},
    {"id": 2, "name": "Jane Doe", "email": "jane@email.com"}
  ],
  "rowCount": 2
}
```

YAML

```
dremio --output yaml sql execute "SELECT * FROM customers LIMIT 2"
```

Output:

```
rows:
  - id: 1
    name: John Doe
    email: john@email.com
  - id: 2
    name: Jane Doe
    email: jane@email.com
rowCount: 2
```

Tips

Use files for complex queries: Store reusable queries in files

```
dremio sql execute --file queries/monthly_report.sql
```

Validate before executing: Catch syntax errors early

```
dremio sql validate --file query.sql && dremio sql execute --file query.sql
```

Use async for long queries: Don't block on large queries

```
dremio sql execute "SELECT * FROM huge_table" --async
```

Set context to simplify queries: Avoid repeating paths

```
dremio sql execute "SELECT * FROM table" --context "MySpace"
```

Export results for analysis: Save to files for further processing

```
dremio sql execute "SELECT * FROM data" --output-file data.json
```

Error Handling

Syntax Error

```
$ dremio sql execute "SELECT * FORM table"
Error: SQL syntax error: Encountered "FORM" at line 1, column 10
```

Solution: Fix the SQL syntax:

```
dremio sql execute "SELECT * FROM table"
```

Table Not Found

```
$ dremio sql execute "SELECT * FROM nonexistent"
Error: Table 'nonexistent' not found
```

Solution: Verify table exists:

```
dremio catalog list | grep "nonexistent"
```

Job Still Running

```
$ dremio sql execute "SELECT * FROM large_table"
⚠ Could not fetch results: Job may still be running
```

Solution: Use async mode or check job status:

```
dremio sql execute "SELECT * FROM large_table" --async
dremio job get <job-id>
```

Platform Differences

Software

Full SQL support via `/api/v3/sql`

Explain and validate work

- All features available

Cloud

- SELECT queries are fully supported via API
- DDL/DML operations are supported but may have limitations compared to Software
- Uses specialized generic SQL endpoint

Best Practices

Validate queries before execution: Catch errors early

Use explain for optimization: Understand query plans

Store queries in files: Version control and reusability

Use async for long queries: Better resource management

Set appropriate context: Simplify query writing

Export results for analysis: Enable downstream processing

Monitor job status: Track query execution

Use limits during development: Test with small datasets first

Advanced Usage

Parameterized Queries

```
# Create template
cat > query_template.sql <<EOF
SELECT * FROM customers WHERE region = '{REGION}' AND created_at >= '{DATE}'
EOF

# Replace parameters and execute
REGION="US"
DATE="2024-01-01"
sed "s/{REGION}/$REGION/g; s/{DATE}/$DATE/g" query_template.sql | dremio sql execute
--file /dev/stdin
```

Query Pipeline

```
# Extract
dremio sql execute "SELECT * FROM source_table" --output-file extracted.json

# Transform (using jq)
cat extracted.json | jq '.rows[] | {id, name, email}' > transformed.json
```

```
# Load (create view with results)
dremio view create --path "Processed.customers" --sql "SELECT * FROM transformed_data"
```

Monitoring and Alerts

```
#!/bin/bash
# monitor_query.sh

# Execute query
RESULT=$(dremio sql execute "SELECT COUNT(*) as count FROM errors WHERE created_at >
NOW() - INTERVAL '1 hour'")

# Parse result
ERROR_COUNT=$(echo $RESULT | jq -r '.rows[0].count')

# Alert if threshold exceeded
if [ $ERROR_COUNT -gt 100 ]; then
  echo "High error count: $ERROR_COUNT" | mail -s "Alert" admin@company.com
fi
```

Source: tables.md

Table Operations

This guide covers table operations for managing physical datasets in Dremio.

Overview

Table Operations allow you to promote datasets to physical datasets (tables), configure file formats, and update table metadata.

Commands

Promote Dataset

Promote a dataset to a physical dataset (table).

```
dremio table promote <DATASET_ID>
```

Arguments:

`DATASET_ID` - The dataset ID (UUID)

Examples:

```
# Promote a dataset to table
dremio table promote abc-123-def-456
```

Configure Format

Configure the file format for a physical dataset.

```
dremio table format <DATASET_ID> --type <FORMAT> [--from-file <FILE>]
```

Arguments:

`DATASET_ID` - The dataset ID (UUID)

Options:

`type` - Format type: CSV, JSON, Parquet, etc. (required)

`from-file` - Load format configuration from JSON file

Examples:

```
# Set CSV format
dremio table format abc-123 --type CSV

# Set format with configuration file
dremio table format abc-123 --type CSV --from-file csv_format.json

# Set JSON format
dremio table format abc-123 --type JSON
```

Update Table

Update table metadata.

```
dremio table update <DATASET_ID> --from-file <FILE>
```

Arguments:

`DATASET_ID` - The dataset ID (UUID)

Options:

`from-file` - Updated table JSON file (required)

Examples:

```
# Update table metadata
dremio table update abc-123 --from-file updated_table.json
```

Format Configuration Examples

CSV Format

```
{
  "type": "CSV",
  "fieldDelimiter": ",",
  "lineDelimiter": "\n",
  "quote": "\"",
  "escape": "\\",
  "skipFirstLine": true,
  "extractHeader": true
}
```

JSON Format

```
{
  "type": "JSON"
}
```

Parquet Format

```
{
  "type": "Parquet",
  "autoCorrectCorruptDates": true
}
```

Scenarios

Promoting and Configuring a CSV File

```
# 1. Get dataset ID
DATASET_ID=$(dremio --output json catalog get-by-path "MySource.data.customers.csv" | jq
-r '.id')

# 2. Promote to table
dremio table promote $DATASET_ID

# 3. Configure CSV format
cat > csv_format.json <<EOF
{
  "type": "CSV",
  "fieldDelimiter": ",",
  "skipFirstLine": true,
  "extractHeader": true
}
```

```
EOF

dremio table format $DATASET_ID --type CSV --from-file csv_format.json
```

Working with JSON Files

```
# Get JSON file dataset
DATASET_ID=$(dremio --output json catalog get-by-path "MySource.data.events.json" | jq
-r '.id')

# Promote and set format
dremio table promote $DATASET_ID
dremio table format $DATASET_ID --type JSON
```

Common Workflows

1. Bulk Dataset Promotion

```
#!/bin/bash
# promote_all_csv.sh - Promote all CSV files in a source

SOURCE="MySource"

# Find all CSV files
dremio --output json catalog list | jq -r ".data[] | select(.path[0] == \"$SOURCE\" and
(.path[-1] | endswith(\".csv\"))) | .id" | while read dataset_id; do
  echo "Promoting: $dataset_id"
  dremio table promote $dataset_id
  dremio table format $dataset_id --type CSV --from-file csv_format.json
done
```

2. Format Configuration Templates

```
#!/bin/bash
# apply_format.sh - Apply format template

DATASET_ID=$1
FORMAT_TYPE=$2

case $FORMAT_TYPE in
  csv)
    cat > format.json <<EOF
{
  "type": "CSV",
  "fieldDelimiter": ",",
  "skipFirstLine": true,
  "extractHeader": true
```

```
}
EOF
    ;;
  tsv)
    cat > format.json <<EOF
{
  "type": "CSV",
  "fieldDelimiter": "\t",
  "skipFirstLine": true,
  "extractHeader": true
}
EOF
    ;;
  json)
    cat > format.json <<EOF
{
  "type": "JSON"
}
EOF
    ;;
esac

dremio table format $DATASET_ID --type ${FORMAT_TYPE^^} --from-file format.json
rm format.json
```

Tips

Promote before formatting: Always promote datasets before configuring format

```
dremio table promote $ID
dremio table format $ID --type CSV
```

Test format settings: Verify format with a query

```
dremio sql execute "SELECT * FROM dataset LIMIT 10"
```

Use format files: Store format configurations for reuse

```
dremio table format $ID --type CSV --from-file standard_csv.json
```

Error Handling

Dataset Already Promoted

```
$ dremio table promote abc-123
Error: Dataset is already a physical dataset
```

Solution: Skip promotion, proceed with format configuration.

Invalid Format Configuration

```
$ dremio table format abc-123 --type CSV --from-file bad_format.json
Error: Invalid format configuration
```

Solution: Verify JSON format and required fields.

Platform Differences

Software

- Full table operations support
- All format types available
- Promotion and format configuration

Cloud

- Table operations available
- Format types may vary
- Project-scoped operations

Best Practices

Promote systematically: Promote datasets as part of source setup

Document formats: Keep format configurations in version control

Test configurations: Verify format settings with sample queries

Use templates: Standardize format configurations

Automate promotion: Script bulk dataset promotion

Format Types Reference

- **CSV** - Comma-separated values
- **TSV** - Tab-separated values
- **JSON** - JSON documents
- **Parquet** - Columnar format
- **Avro** - Row-based format

Excel - Excel spreadsheets

Summary

Promote: Convert datasets to physical datasets

Format: Configure file format settings

Update: Modify table metadata

Automate: Use scripts for bulk operations

Test: Verify format with queries

Source: tags-wiki.md

Tag and Wiki Management

This guide covers tag and wiki management for documenting and organizing catalog objects in Dremio.

Overview

Tags and **Wiki** provide collaboration features:

Tags: Labels for categorizing and organizing datasets (views and tables only)

Wiki: Markdown documentation for any catalog object

Tag Commands

Set Tags

Set tags on a catalog object (views and tables only).

```
dremio tag set <CATALOG_ID> --tags <TAGS>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Options:

`tags TEXT` - Comma-separated list of tags (required)

Examples:

```
# Set single tag
dremio tag set abc-123 --tags analytics
```

```
# Set multiple tags
dremio tag set abc-123 --tags analytics,production,sensitive

# Set tags with spaces
dremio tag set abc-123 --tags "customer data,pii,gdpr compliant"
```

Get Tags

Retrieve tags from a catalog object.

```
dremio tag get <CATALOG_ID>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Examples:

```
# Get tags
dremio tag get abc-123

# Get in JSON format
dremio --output json tag get abc-123
```

Delete Tags

Remove all tags from a catalog object.

```
dremio tag delete <CATALOG_ID>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Examples:

```
# Delete tags (with confirmation)
dremio tag delete abc-123

# Delete without confirmation
dremio tag delete abc-123 --yes
```

Wiki Commands

Set Wiki

Set wiki documentation on a catalog object.

```
dremio wiki set <CATALOG_ID> --text <TEXT>
dremio wiki set <CATALOG_ID> --file <FILE>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Options:

`text TEXT` - Wiki markdown text

`file PATH` - Load wiki from file

Examples:

```
# Set wiki with inline text
dremio wiki set abc-123 --text "# My Dataset\n\nThis dataset contains customer information."

# Set wiki from file
dremio wiki set abc-123 --file README.md

# Set comprehensive wiki
cat > dataset_wiki.md <<EOF
# Customer Dataset

## Overview
This dataset contains customer information for analytics.

## Schema
- id: Customer ID
- name: Customer name
- email: Customer email
- region: Geographic region

## Usage
Use this dataset for customer segmentation and analysis.

## Owners
- Data Team: data@company.com
EOF

dremio wiki set abc-123 --file dataset_wiki.md
```

Get Wiki

Retrieve wiki documentation from a catalog object.

```
dremio wiki get <CATALOG_ID> [OPTIONS]
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Options:

`output-file PATH` - Save wiki to file

Examples:

```
# Get wiki
dremio wiki get abc-123

# Save wiki to file
dremio wiki get abc-123 --output-file README.md

# Get in JSON format
dremio --output json wiki get abc-123
```

Delete Wiki

Remove wiki documentation from a catalog object.

```
dremio wiki delete <CATALOG_ID>
```

Arguments:

`CATALOG_ID` - The catalog object ID (UUID)

Examples:

```
# Delete wiki (with confirmation)
dremio wiki delete abc-123

# Delete without confirmation
dremio wiki delete abc-123 --yes
```

Scenarios

Documenting a View

```
# 1. Create a view
dremio view create --path "Analytics.customer_summary" --sql "SELECT * FROM customers"

# 2. Get view ID
VIEW_ID=$(dremio --output json view get-by-path "Analytics.customer_summary" | jq -r
'.id')

# 3. Add tags
dremio tag set $VIEW_ID --tags "analytics,customer-data,production"
```

```
# 4. Add wiki documentation
cat > view_docs.md <<EOF
# Customer Summary View

## Purpose
Provides a summary of customer data for analytics dashboards.

## Source
- Base table: customers
- Refresh: Daily at 2 AM UTC

## Columns
- customer_id: Unique identifier
- name: Customer name
- total_orders: Lifetime order count
- total_spent: Lifetime revenue

## Usage Examples
\`\`\`sql
-- Top customers by revenue
SELECT * FROM Analytics.customer_summary
ORDER BY total_spent DESC
LIMIT 10
\`\`\`

## Owners
- Analytics Team: analytics@company.com
EOF

dremio wiki set $VIEW_ID --file view_docs.md

# 5. Verify
dremio tag get $VIEW_ID
dremio wiki get $VIEW_ID
```

Organizing with Tags

```
# Tag datasets by environment
dremio tag set dev-view-id --tags development,testing
dremio tag set staging-view-id --tags staging,pre-production
dremio tag set prod-view-id --tags production,critical

# Tag by data classification
dremio tag set customer-view-id --tags pii,sensitive,gdpr
dremio tag set public-view-id --tags public,non-sensitive

# Tag by team ownership
dremio tag set sales-view-id --tags sales-team,revenue
dremio tag set marketing-view-id --tags marketing-team,campaigns
```

Documentation Workflow

```
# 1. Create documentation template
cat > template.md <<EOF
# {DATASET_NAME}

## Overview
{DESCRIPTION}

## Schema
{SCHEMA_INFO}

## Usage
{USAGE_EXAMPLES}

## Owners
{OWNER_INFO}

## Last Updated
{DATE}
EOF

# 2. Fill in template for each dataset
sed "s/{DATASET_NAME}/Customer Data/g; s/{DESCRIPTION}/Customer information/g"
template.md > customer_wiki.md

# 3. Apply to datasets
dremio wiki set <customer-view-id> --file customer_wiki.md

# 4. Export all wikis for backup
for id in $(dremio --output json view list | jq -r '.[].id'); do
  dremio wiki get $id --output-file "wikis/${id}.md"
done
```

Common Workflows

1. Data Governance

```
# Tag sensitive datasets
SENSITIVE_VIEWS=$(dremio --output json view list | jq -r '.[] | select(.path[] |
contains("customer")) | .id')

for view_id in $SENSITIVE_VIEWS; do
  dremio tag set $view_id --tags "pii,sensitive,restricted"
done

# Add compliance documentation
for view_id in $SENSITIVE_VIEWS; do
    dremio wiki set $view_id --text "# Data Classification\n\n**Classification**:
Sensitive\n**Compliance**: GDPR, CCPA\n**Access**: Restricted to authorized personnel
only"
```

```
done
```

2. Dataset Catalog

```
# Create comprehensive catalog
dremio --output json view list | jq -r '.[] | .id' | while read view_id; do
  # Get view details
  VIEW=$(dremio --output json view get $view_id)
  NAME=$(echo $VIEW | jq -r '.path | join(".")')

  # Create documentation
  cat > "catalog/${view_id}.md" <<EOF
# $NAME

## Tags
$(dremio tag get $view_id)

## Wiki
$(dremio wiki get $view_id)

## SQL
\`\`\`sql
$(echo $VIEW | jq -r '.sql')
\`\`\`
EOF
done
```

3. Migration Documentation

```
# Export tags and wikis before migration
mkdir -p migration/tags migration/wikis

dremio --output json view list | jq -r '.[] | .id' | while read id; do
  dremio --output json tag get $id > "migration/tags/${id}.json"
  dremio wiki get $id --output-file "migration/wikis/${id}.md"
done

# After migration, restore
for id_file in migration/tags/*.json; do
  id=$(basename $id_file .json)
  tags=$(cat $id_file | jq -r '.tags | join(",")')
  dremio tag set $id --tags "$tags"
  dremio wiki set $id --file "migration/wikis/${id}.md"
done
```

Tips

Use consistent tag naming: Establish conventions

```
# Good: lowercase, hyphenated
dremio tag set $id --tags "customer-data,production,pii"

# Avoid: mixed case, spaces
dremio tag set $id --tags "Customer Data,PRODUCTION,PII"
```

- **Document in Markdown**: Use proper formatting

```
# Dataset Name

## Overview
Brief description

## Schema
| Column | Type | Description |
|--------|------|-------------|
| id     | INT  | Primary key |

## Examples
\`\`\`sql
SELECT * FROM dataset LIMIT 10
\`\`\`
```

- **Version control wikis**: Store in git

```
dremio wiki get $id --output-file docs/datasets/my_dataset.md
git add docs/datasets/my_dataset.md
git commit -m "Update dataset documentation"
```

- **Automate tagging**: Use scripts for consistency

```
# Tag all views in Analytics space
dremio --output json view list --space Analytics | jq -r '.[].id' | \
  xargs -I {} dremio tag set {} --tags "analytics,production"
```

Important Notes

Tag Limitations

⚠ **Tags can only be set on views and tables**, not on:

- Spaces
- Folders
- Sources

Attempting to tag other objects will result in:

```
Error: Labels may only be set on views and tables
```

Wiki Support

Wiki can be set on any catalog object:

- Spaces
- Folders
- Views
- Tables
- Sources

Error Handling

Cannot Tag Spaces

```
$ dremio tag set space-id --tags analytics
Error: Labels may only be set on views and tables
```

Solution: Only tag views and tables:

```
# Get view ID instead
VIEW_ID=$(dremio --output json view get-by-path "MySpace.MyView" | jq -r '.id')
dremio tag set $VIEW_ID --tags analytics
```

Object Not Found

```
$ dremio tag get invalid-id
Error: Resource not found
```

Solution: Verify the object ID:

```
dremio catalog get-by-path "MySpace.MyView"
```

Platform Differences

Software

- Full tag and wiki support
- Tags work on views and tables
- Wiki works on all objects

Cloud

- Full tag and wiki support
- Same limitations as Software
- Project-scoped endpoints

Best Practices

1. **Establish tagging conventions**: Define standard tags
2. **Document all production datasets**: Add wikis to important views
3. **Use tags for governance**: Mark sensitive data
4. **Version control documentation**: Store wikis in git
5. **Automate tagging**: Script common patterns
6. **Regular audits**: Review and update documentation
7. **Team ownership**: Assign dataset owners in wiki
8. **Include examples**: Add SQL examples in wikis

Advanced Usage

Bulk Tagging

```
#!/bin/bash
# bulk_tag.sh - Tag multiple datasets

TAG_LIST="analytics,production,verified"

# Tag all views in a space
dremio --output json view list --space Analytics | jq -r '.[].id' | \
while read view_id; do
  echo "Tagging $view_id..."
  dremio tag set $view_id --tags "$TAG_LIST"
done
```

Documentation Generator

```
#!/bin/bash
# generate_docs.sh - Auto-generate documentation

VIEW_ID=$1

# Get view details
VIEW=$(dremio --output json view get $VIEW_ID)
```

```
NAME=$(echo $VIEW | jq -r '.path | join(".")')
SQL=$(echo $VIEW | jq -r '.sql')

# Generate wiki
cat > wiki.md <<EOF
# $NAME

## SQL Definition
\`\`\`sql
$SQL
\`\`\`

## Created
$(date)

## Owner
Data Team

## Usage
This view is used for analytics and reporting.
EOF

# Set wiki
dremio wiki set $VIEW_ID --file wiki.md
```

Tag-Based Search

```
# Find all production datasets
dremio --output json view list | jq -r '.[] | .id' | while read id; do
  TAGS=$(dremio --output json tag get $id 2>/dev/null | jq -r '.tags[]' 2>/dev/null)
  if echo "$TAGS" | grep -q "production"; then
    echo "Production dataset: $id"
  fi
done
```

Summary

- **Tags**: Categorize views and tables
- **Wiki**: Document any catalog object
- **Markdown**: Use rich formatting in wikis
- **Governance**: Use tags for data classification
- **Automation**: Script tagging and documentation
- **Version Control**: Store wikis in git

Source: users.md

User Management

This guide covers user management operations for administering user accounts in Dremio.

Overview

User Management allows administrators to create, update, and manage user accounts. This is primarily available in Dremio Software.

Commands

List Users

```
dremio user list
```

Get User

```
dremio user get <USER_ID>
```

Create User

```
dremio user create --name "John Doe" --email john@company.com [--username john] [--password secret]
dremio user create --from-file user.json
```

Update User

```
dremio user update <USER_ID> --from-file updated_user.json
```

Delete User

```
dremio user delete <USER_ID>
```

Examples

```
# List all users
dremio user list

# Create user
dremio user create --name "Jane Analyst" --email jane@company.com
```

```
# Get user details
dremio user get user-123

# Delete user
dremio user delete user-123
```

User File Format

```
{
  "name": "John Doe",
  "email": "john@company.com",
  "userName": "john",
  "password": "initial_password"
}
```

Notes

- User management requires administrative privileges
- Primarily available in Dremio Software
- Cloud has different user management (via cloud console)

Source: views.md

View Management

This guide covers view management operations including creating, updating, and managing virtual datasets (views) in Dremio.

Commands

Create View

Create a new view with a SQL query.

```
dremio view create --path <PATH> --sql <SQL> [OPTIONS]
dremio view create --from-file <FILE>
```

Options:

- `path TEXT` - View path as JSON array or dot-separated (required unless using `from-file`)
- `sql TEXT` - SQL query for the view

`from-file PATH` - Load view definition from JSON file

Examples:

```
# Create simple view
dremio view create \
  --path '["MySpace", "MyView"]' \
  --sql "SELECT * FROM customers WHERE active = true"

# Create with dot-separated path
dremio view create \
  --path "Analytics.active_customers" \
  --sql "SELECT * FROM customers WHERE active = true"

# Create from file
cat > view.json <<EOF
{
  "entityType": "dataset",
  "type": "VIRTUAL_DATASET",
  "path": ["Analytics", "monthly_sales"],
   "sql": "SELECT DATE_TRUNC('month', order_date) as month, SUM(amount) as total FROM
orders GROUP BY 1"
}
EOF
dremio view create --from-file view.json

# Create complex view
dremio view create \
  --path "Reports.customer_summary" \
   --sql "SELECT c.id, c.name, COUNT(o.id) as order_count, SUM(o.amount) as total_spent
FROM customers c LEFT JOIN orders o ON c.id = o.customer_id GROUP BY c.id, c.name"
```

Get View

Retrieve view details by ID.

```
dremio view get <VIEW_ID> [OPTIONS]
```

Arguments:

`VIEW_ID` - The view ID (UUID)

Options:

`include TEXT` - Include additional fields (e.g., `sql`, `permissions`)

Examples:

```
# Get view details
dremio view get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3

# Get view with SQL
dremio view get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --include sql
```

```
# Get in JSON format
dremio --output json view get 4cc92138-34e8-4c84-ad03-abfb23b6d5f3
```

Get View by Path

Retrieve view details by path.

```
dremio view get-by-path <PATH> [OPTIONS]
```

Arguments:

`PATH` - The view path (dot-separated or slash-separated)

Options:

`include TEXT` - Include additional fields

Examples:

```
# Get by dot-separated path
dremio view get-by-path "Analytics.monthly_sales"

# Get by slash-separated path
dremio view get-by-path "Analytics/Reports/summary"

# Get with SQL definition
dremio view get-by-path "Analytics.monthly_sales" --include sql
```

Update View

Update an existing view's SQL or definition.

```
dremio view update <VIEW_ID> --sql <SQL>
dremio view update <VIEW_ID> --from-file <FILE>
```

Arguments:

`VIEW_ID` - The view ID (UUID)

Options:

`sql TEXT` - New SQL query for the view

`from-file PATH` - Load updated definition from JSON file

Examples:

```
# Update view SQL
dremio view update 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 \
  --sql "SELECT * FROM customers WHERE active = true AND created_at > '2024-01-01'"
```

```
# Update from file
cat > updated_view.json <<EOF
{
  "entityType": "dataset",
  "type": "VIRTUAL_DATASET",
  "id": "4cc92138-34e8-4c84-ad03-abfb23b6d5f3",
  "path": ["Analytics", "monthly_sales"],
    "sql":  "SELECT  DATE_TRUNC('month',  order_date)  as  month,  SUM(amount)  as  total,
COUNT(*) as count FROM orders GROUP BY 1"
}
EOF
dremio view update 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --from-file updated_view.json
```

Delete View

Delete a view.

```
dremio view delete <VIEW_ID> [OPTIONS]
```

Arguments:

`VIEW_ID` - The view ID (UUID)

Options:

`tag TEXT` - Version tag for optimistic concurrency control

Examples:

```
# Delete view (with confirmation)
dremio view delete 4cc92138-34e8-4c84-ad03-abfb23b6d5f3

# Delete without confirmation
dremio view delete 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --yes

# Delete with specific tag
dremio view delete 4cc92138-34e8-4c84-ad03-abfb23b6d5f3 --tag "version-tag-123"
```

List Views

List all views in the catalog.

```
dremio view list [OPTIONS]
```

Options:

`space TEXT` - Filter views by space name

Examples:

```
# List all views
dremio view list

# List views in specific space
dremio view list --space Analytics

# List in JSON format
dremio --output json view list
```

Scenarios

Creating a Data Mart

```
# 1. Create base views
dremio view create \
  --path "DataMart.dim_customers" \
  --sql "SELECT id, name, email, created_at FROM raw.customers"

dremio view create \
  --path "DataMart.dim_products" \
  --sql "SELECT id, name, category, price FROM raw.products"

dremio view create \
  --path "DataMart.fact_orders" \
  --sql "SELECT id, customer_id, product_id, amount, order_date FROM raw.orders"

# 2. Create summary view
dremio view create \
  --path "DataMart.sales_summary" \
   --sql "SELECT c.name as customer, p.name as product, SUM(o.amount) as total FROM
DataMart.fact_orders o JOIN DataMart.dim_customers c ON o.customer_id = c.id JOIN
DataMart.dim_products p ON o.product_id = p.id GROUP BY 1, 2"
```

Iterative View Development

```
# 1. Create initial view
dremio view create \
  --path "Analytics.sales" \
  --sql "SELECT * FROM orders"

# 2. Test the view
dremio sql execute "SELECT * FROM Analytics.sales LIMIT 10"

# 3. Get view ID
VIEW_ID=$(dremio --output json view get-by-path "Analytics.sales" | jq -r '.id')

# 4. Update with filters
dremio view update $VIEW_ID \
```

```
  --sql "SELECT * FROM orders WHERE order_date >= '2024-01-01'"

# 5. Test again
dremio sql execute "SELECT COUNT(*) FROM Analytics.sales"

# 6. Add aggregations
dremio view update $VIEW_ID \
  --sql "SELECT DATE_TRUNC('day', order_date) as day, SUM(amount) as total FROM orders
WHERE order_date >= '2024-01-01' GROUP BY 1"
```

View Migration

```
# 1. Export view from source
dremio --profile source --output json view get-by-path "Analytics.summary" >
view_export.json

# 2. Modify for target environment
cat view_export.json | jq '.path = ["NewAnalytics", "summary"]' > view_import.json

# 3. Create in target
dremio --profile target view create --from-file view_import.json
```

View Documentation

```
# Export all views with SQL
dremio --output json view list | jq '.[] | {path: .path, sql: .sql}' >
view_documentation.json

# Generate markdown documentation
cat view_documentation.json | jq -r '.[] | "## \(.path |
join("."))\n\n```sql\n\(.sql)\n```\n"' > views.md
```

Common Workflows

1. Create View Hierarchy

```
# Level 1: Raw data views
dremio view create --path "Bronze.customers" --sql "SELECT * FROM source.customers"
dremio view create --path "Bronze.orders" --sql "SELECT * FROM source.orders"

# Level 2: Cleaned data views
dremio view create --path "Silver.customers" --sql "SELECT id, TRIM(name) as name,
LOWER(email) as email FROM Bronze.customers WHERE id IS NOT NULL"

# Level 3: Business logic views
dremio view create --path "Gold.customer_metrics" --sql "SELECT c.id, c.name,
COUNT(o.id) as order_count, SUM(o.amount) as lifetime_value FROM Silver.customers c LEFT
```

```
JOIN Bronze.orders o ON c.id = o.customer_id GROUP BY c.id, c.name"
```

2. View Versioning

```
# Create v1
dremio view create --path "Analytics.metrics_v1" --sql "SELECT * FROM data"

# Create v2 with improvements
dremio view create --path "Analytics.metrics_v2" --sql "SELECT *, additional_field FROM
data"

# Update production view to v2
VIEW_ID=$(dremio --output json view get-by-path "Analytics.metrics" | jq -r '.id')
dremio view update $VIEW_ID --sql "SELECT * FROM Analytics.metrics_v2"
```

3. View Testing

```
# Create test view
dremio view create --path "Testing.new_metric" --sql "SELECT customer_id, SUM(amount) as
total FROM orders GROUP BY customer_id"

# Test with sample data
dremio sql execute "SELECT * FROM Testing.new_metric LIMIT 10"

# Validate results
dremio sql execute "SELECT COUNT(*), SUM(total) FROM Testing.new_metric"

# Promote to production
dremio view create --path "Production.customer_totals" --sql "SELECT customer_id,
SUM(amount) as total FROM orders GROUP BY customer_id"

# Delete test view
VIEW_ID=$(dremio --output json view get-by-path "Testing.new_metric" | jq -r '.id')
dremio view delete $VIEW_ID --yes
```

Tips

Use meaningful names: Make view paths descriptive

```
dremio view create --path "Analytics.monthly_revenue_by_region" --sql "..."
```

Document complex SQL: Add comments in SQL

```
-- Calculate customer lifetime value
SELECT
  c.id,
  c.name,
```

```
  SUM(o.amount) as ltv
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.name
```

Test before updating: Always test SQL before updating production views

```
dremio sql execute "SELECT * FROM (YOUR_NEW_SQL) LIMIT 10"
```

Use version control: Store view definitions in git

```
dremio --output json view get-by-path "Analytics.summary" > views/analytics_summary.json
git add views/analytics_summary.json
git commit -m "Update analytics summary view"
```

Error Handling

View Already Exists

```
$ dremio view create --path "Analytics.summary" --sql "SELECT 1"
Error: View already exists
```

Solution: Update instead of create:

```
VIEW_ID=$(dremio --output json view get-by-path "Analytics.summary" | jq -r '.id')
dremio view update $VIEW_ID --sql "SELECT 1"
```

Invalid SQL

```
$ dremio view create --path "Analytics.bad" --sql "SELECT * FORM table"
Error: SQL syntax error
```

Solution: Test SQL first:

```
dremio sql execute "SELECT * FROM table LIMIT 1"
```

Path Not Found

```
$ dremio view create --path "NonExistent.view" --sql "SELECT 1"
Error: Parent path does not exist
```

Solution: Create parent space/folder first:

```
dremio space create --name "NonExistent"
dremio view create --path "NonExistent.view" --sql "SELECT 1"
```

Platform Differences

Cloud

- Views created in project catalog
- Path: `source.namespace.view`

Software

- Views created in spaces or catalog
- Path: `space.view` or `catalog.namespace.view`

Best Practices

- **Organize views logically**: Use spaces/folders for organization
- **Keep SQL readable**: Format and comment complex queries
- **Test thoroughly**: Validate views before production use
- **Version control**: Track view definitions in git
- **Document dependencies**: Note which views depend on others
- **Use consistent naming**: Follow naming conventions
- **Clean up unused views**: Delete obsolete views regularly