Salesforce CLI Command Reference

Salesforce, Spring '25

Last updated: May 2, 2025

CONTENTS

SALESFORCE CLI COMMAND REFERENCE

This command reference contains information about the Salesforce CLI commands and their flags.

Salesforce CLI Release Notes

Use the Release Notes to learn about the most recent updates and changes to Salesforce CLI.

We release new versions of Salesforce CLI weekly. Read the weekly release notes to learn about new features, changes, and bug fixes in both the current release and the release candidate.

sf

Commands to manage Salesforce DX projects, create and manage scratch orgs and sandboxes, synchronize source to and from orgs, create and install packages, and more.

This version of the `sf` command reference includes details about version 2.87.6 of the `sf` executable of Salesforce CLI and the following plug-in versions:

- `@salesforce/plugin-deploy-retrieve` version 3.22.5
- `@salesforce/plugin-settings` version 2.4.25
- `@salesforce/plugin-info` version 3.4.56
- `@salesforce/plugin-sobject` version 1.4.55
- `@salesforce/plugin-limits` version 3.3.52
- `@salesforce/plugin-schema` version 3.3.59
- `@salesforce/plugin-custom-metadata` version 3.3.51

- `@salesforce/plugin-data` version 4.0.29
- `@salesforce/plugin-community` version 3.3.21
- `@salesforce/plugin-signups` version 2.6.23
- `@salesforce/plugin-user` version 3.6.19
- `@salesforce/plugin-org` version 5.7.3
- `@salesforce/plugin-packaging` version 2.13.2
- `@salesforce/plugin-templates` version 56.3.45
- `@salesforce/plugin-apex` version 3.6.15
- `@salesforce/plugin-auth` version 3.6.115
- `@salesforce/plugin-dev` version 2.5.1
- `@salesforce/sfdx-plugin-lwc-test` version 1.2.1
- `@salesforce/plugin-devops-center` version 1.2.27
- `@salesforce/plugin-marketplace` version 1.3.8
- `@salesforce/sfdx-scanner` version 4.12.0
- `@salesforce/plugin-code-analyzer` version 5.0.0
- `@salesforce/plugin-api` version 1.3.3
- `@salesforce/plugin-agent` version 1.20.12
- `@salesforce/plugin-flow` version 1.0.2

For information about installing Salesforce CLI, see the *Salesforce CLI Setup Guide*.

For information about Salesforce CLI changes, see the *Salesforce CLI Release Notes*.

`agent` Commands

Commands to work with agents.

`agent create` (Beta)

Create an agent in your org using a local agent spec file.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent create`

To run this comand, you must have an agent spec file, which is a YAML file that define the agent properties and contains a list of AI-generated topics. Topics define the range of jobs the agent can handle. Use the "agent generate agent-spec" CLI command to generate an agent spec file. Then specify the file to this command using the --spec flag, along with the name (label) of the new agent with the --agent-name flag. If you don't specify any of the required flags, the command prompts you.

When this command completes, your org contains the new agent, which you can then edit and customize in the Agent Builder UI. The new agent's topics are the same as the ones listed in the agent spec file. The agent might also have some AI-generated actions, or you can add them. This command also retrieves all the metadata files associated with the new agent to your local Salesforce DX project.

Use the --preview flag to review what the agent looks like without actually saving it in your org. When previewing, the command creates a JSON file in the current directory with all the agent details. The name of the JSON file is the agent's API name and a timestamp.

To open the new agent in your org's Agent Builder UI, run this command: "sf org open agent --name <agent-api-name>".

Examples for `agent create`

Create an agent by being prompted for the required information, such as the agent spec file and agent name, and then create it in your default org:

```
sf agent create
```

Create an agent by specifying the agent name, API name, and spec file with flags; use the org with alias "my-org"; the command fails if the API name is already being used in your org:

```
sf agent create --agent-name "Resort Manager" --agent-api-name Resort_Manager --spec
specs/resortManagerAgent.yaml --target-org my-org
```

Preview the creation of an agent named "Resort Manager" and use your default org:

```
sf agent create --agent-name "Resort Manager" --spec specs/resortManagerAgent.yaml --preview
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`--agent-name AGENT-NAME`
Optional

Name (label) of the new agent.

Type: option

`--agent-api-name AGENT-API-NAME`
Optional

API name of the new agent; if not specified, the API name is derived from the agent name (label); the API name must not exist in the org.

Type: option

`--spec SPEC`
Optional

Path to an agent spec file.

Type: option

`--preview`

Optional

Preview the agent without saving it in your org.

Type: boolean

`agent generate agent-spec` (Beta)

Generate an agent spec, which is a YAML file that captures what an agent can do.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent generate agent-spec`

The first step in creating an agent in your org with Salesforce CLI is to generate an agent spec using this command. An agent spec is a YAML-formatted file that contains information about the agent, such as its role and company description, and then an AI-generated list of topics based on this information. Topics define the range of jobs your agent can handle.

Use flags, such as --role and --company-description, to provide details about your company and the role that the agent plays in your company. If you prefer, you can also be prompted for the basic information; use --full-interview to be prompted for all required and optional properties. Upon command execution, the large language model (LLM) associated with your org uses the provided information to generate a list of topics for the agent. Because the LLM uses the company and role information to generate the topics, we recommend that you provide accurate, complete, and specific details so the LLM generates the best and most relevant topics. Once generated, you can edit the spec file; for example, you can remove topics that don't apply or change a topic's description.

You can also iterate the spec generation process by using the --spec flag to pass an existing agent spec file to this command, and then using the --role, --company-description, etc, flags to refine your agent properties. Iteratively improving the description of your agent allows the LLM to generate progressively better topics.

You can also specify other agent properties, such as a custom prompt template, how to ground the prompt template to add context to the agent's prompts, the tone of the prompts, and the username of a user in the org to assign to the agent.

When your agent spec is ready, you then create the agent in your org by running the "agent create" CLI command and specifying the spec with the --spec flag.

Examples for `agent generate agent-spec`

Generate an agent spec in the default location and use flags to specify the agent properties, such as its role and your company details; use your default org:

```
sf agent generate agent-spec --type customer --role "Field customer complaints and manage
 employee schedules." --company-name "Coral Cloud Resorts" --company-description "Provide
 customers with exceptional destination activities, unforgettable experiences, and
reservation services."
```

Generate an agent spec by being prompted for the required agent properties and generate a maxiumum of 5 topics; write the generated file to the "specs/resortManagerSpec.yaml" file and use the org with alias "my-org":

```
sf agent generate agent-spec --max-topics 5 --output-file specs/resortManagerAgent.yaml
--target-org my-org
```

Be prompted for all required and optional agent properties; use your default org:

```
sf agent generate agent-spec --full-interview
```

Specify an existing agent spec file called "specs/resortManagerAgent.yaml", and then overwrite it with a new version that contains newly AI-generated topics based on the updated role information passed in with the --role flag:

```
sf agent generate agent-spec --spec specs/resortManagerAgent.yaml --output-file
specs/resortManagerAgent.yaml --role "Field customer complaints, manage employee schedules,
 and ensure all resort operations are running smoothly"
```

Specify that the conversational tone of the agent is formal and to attach the "resortmanager@myorg.com" username to it; be prompted for the required properties and use your default org:

```
sf agent generate agent-spec --tone formal --agent-user resortmanager@myorg.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

--type TYPE
Optional

Type of agent to create. Internal types are copilots used internally by your company and customer types are the agents you create for your customers.

Type: option

Permissible values are: customer, internal

--role ROLE
Optional

Role of the agent.

Type: option

--company-name COMPANY-NAME
Optional

Name of your company.

Type: option

--company-description COMPANY-DESCRIPTION
Optional

Description of your company.

Type: option

--company-website COMPANY-WEBSITE
Optional

Website URL of your company.

Type: option

--max-topics MAX-TOPICS
Optional

Maximum number of topics to generate in the agent spec; default is 5.

Type: option

--agent-user AGENT-USER
Optional

Username of a user in your org to assign to your agent; determines what your agent can access and do.

Type: option

--enrich-logs ENRICH-LOGS
Optional

Adds agent conversation data to event logs so you can view all agent session activity in one place.

Type: option

Permissible values are: true, false

--tone TONE
Optional

Conversational style of the agent, such as how it expresses your brand personality in its messages through word choice, punctuation, and sentence structure.

Type: option

Permissible values are: formal, casual, neutral

--spec SPEC
Optional

Agent spec file, in YAML format, to use as input to the command.

Type: option

--output-file OUTPUT-FILE
Optional

Path for the generated YAML agent spec file; can be an absolute or relative path.

Type: option

Default value: specs/agentSpec.yaml

--full-interview
Optional

Prompt for both required and optional flags.

Type: boolean

--prompt-template PROMPT-TEMPLATE
Optional

API name of a customized prompt template to use instead of the default prompt template.

Type: option

--grounding-context GROUNDING-CONTEXT
Optional

Context information and personalization that's added to your prompts when using a custom prompt template.

Type: option

--force-overwrite
Optional

Don't prompt the user to confirm that an existing spec file will be overwritten.

Type: boolean

`agent generate template` (Beta)

Generate an agent template from an existing agent in your DX project so you can then package the template in a managed package.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent generate template`

At a high-level, agents are defined by the Bot, BotVersion, and GenAiPlanner metadata types. The GenAiPlanner type in turn defines the agent's topics and actions. This command uses the metadata files for these three types, located in your local DX project, to generate a BotTemplate file for a specific agent (Bot). You then use the BotTemplate file, along with the GenAiPlanner file that references the BotTemplate, to package the template in a managed package that you can share between orgs or on AppExchange.

Use the --agent-file flag to specify the relative or full pathname of the Bot metadata file, such as force-app/main/default/bots/My_Awesome_Agent/My_Awesome_Agent.bot-meta.xml. A single Bot can have multiple BotVersions, so use the --agent-version flag to specify the version. The corresponding BotVersion file must exist locally. For example, if you specify "--agent-version 4", then the file force-app/main/default/bots/My_Awesome_Agent/v4.botVersion-meta.xml must exist.

The new BotTemplate file is generated in the "botTemplates" directory in your local package directory, and has the name <Agent_API_name>_v<Version>_Template.botTemplate-meta.xml, such as

force-app/main/default/botTemplates/My_Awesome_Agent_v4_Template.botTemplate-meta.xml. The command displays the full pathname of the generated files when it completes.

Examples for `agent generate template`

Generate an agent template from a Bot metadata file in your DX project that corresponds to the My_Awesome_Agent agent; use version 1 of the agent.

```
sf agent generate template --agent-file
force-app/main/default/bots/My_Awesome_Agent/My_Awesome_Agent.bot-meta.xml --agent-version
 1
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`--agent-version AGENT-VERSION`
Required

Version of the agent (BotVersion).

Type: option

`-f | --agent-file AGENT-FILE`
Required

Path to an agent (Bot) metadata file.

Type: option

`agent generate test-spec` (Beta)

Generate an agent test spec, which is a YAML file that lists the test cases for testing a specific agent.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent generate test-spec`

The first step when using Salesforce CLI to create an agent test in your org is to use this interactive command to generate a local YAML-formatted test spec file. The test spec YAML file contains information about the agent being tested, such as its API name, and then one or more test cases. This command uses the metadata components in your DX project when prompting for information, such as the agent API name; it doesn't look in your org.

To generate a specific agent test case, this command prompts you for this information; when possible, the command provides a list of options for you to choose from:

- Utterance: Natural language statement, question, or command used to test the agent.

- Expected topic: API name of the topic you expect the agent to use when responding to the utterance.

- Expected actions: One or more API names of the expection actions the agent takes.

- Expected outcome: Natural language description of the outcome you expect.

When your test spec is ready, you then run the "agent test create" command to actually create the test in your org and synchronize the metadata with your DX project. The metadata type for an agent test is AiEvaluationDefinition.

If you have an existing AiEvaluationDefinition metadata XML file in your DX project, you can generate its equivalent YAML test spec file with the --from-definition flag.

Examples for `agent generate test-spec`

Generate an agent test spec YAML file interactively:

```
sf agent generate test-spec
```

Generate an agent test spec YAML file and specify a name for the new file; if the file exists, overwrite it without confirmation:

```
sf agent generate test-spec --output-file specs/Resort_Manager-new-version-testSpec.yaml
--force-overwrite
```

Generate an agent test spec YAML file from an existing AiEvaluationDefinition metadata XML file in your DX project:

```
sf agent generate test-spec --from-definition
force-app//main/default/aiEvaluationDefinitions/Resort_Manager_Tests.aiEvaluationDefinition-meta.xml
```

Flags

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-d | --from-definition FROM-DEFINITION`

Optional

Filepath to the AIEvaluationDefinition metadata XML file in your DX project that you want to convert to a test spec YAML file.

Type: option

`--force-overwrite`

Optional

Don't prompt for confirmation when overwriting an existing test spec YAML file.

Type: boolean

-f | --output-file OUTPUT-FILE

Optional

Name of the generated test spec YAML file. Default value is "specs/<AGENT_API_NAME>-testSpec.yaml".

Type: option

agent preview

Interact with an active agent to preview how the agent responds to your statements, questions, and commands (utterances).

Description for agent preview

Use this command to have a natural language conversation with an active agent in your org, as if you were an actual user. The interface is simple: in the "Start typing..." prompt, enter a statement, question, or command; when you're done, enter Return. Your utterance is posted on the right along with a timestamp. The agent then responds on the left. To exit the conversation, hit ESC or Control+C.

This command is useful to test if the agent responds to your utterances as you expect. For example, you can test that the agent uses a particular topic when asked a question, and then whether it invokes the correct action associated with that topic. This command is the CLI-equivalent of the Conversation Preview panel in your org's Agent Builder UI.

When the session concludes, the command asks if you want to save the API responses and chat transcripts. By default, the files are saved to the "./temp/agent-preview" directory. Specify a new default directory by setting the environment variable "SF_AGENT_PREVIEW_OUTPUT_DIR" to the directory. Or you can pass the directory to the --output-dir flag.

Find the agent's API name in its main details page in your org's Agent page in Setup.

Before you use this command, you must complete these steps:

---

1. Create a connected app in your org as described in the "Create a Connected App" section here: https://developer.salesforce.com/docs/einstein/genai/guide/agent-api-get-started.html#create-a-connected-app. Do these four additional steps:

a. When specifying the connected app's Callback URL, add this second callback URL on a new line: "http://localhost:1717/OauthRedirect".

b. When adding the scopes to the connected app, add "Manage user data via Web browsers (web)".

c. Ensure that the "Require Secret for Web Server Flow" option is not selected.

d. Make note of the user that you specified as the "Run As" user when updating the Client Credentials Flow section.

2. Add the connected app to your agent as described in the "Add Connected App to Agent" section here: https://developer.salesforce.com/docs/einstein/genai/guide/agent-api-get-started.html#add-connected-app-to-agent.

3. Copy the consumer key from your connected app as described in the "Obtain Credentials" section here: https://developer.salesforce.com/docs/einstein/genai/guide/agent-api-get-started.html#obtain-credentials.

4. Set the "SFDX_AUTH_SCOPES" environment variable to "refresh_token sfap_api chatbot_api web api". This step ensures that you get the specific OAuth scopes required by this command.

5. Using the username of the user you specified as the "Run As" user above, authorize your org using the web server flow, as described in this document: https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_web_flow.htm.

IMPORTANT: You must use the "--client-id <CONNECTED-APP-CONSUMER-KEY>" flag of "org login web", where CONNECTED-APP-CONSUMER-KEY is the consumer key you previously copied. This step ensures that the "org login web" command uses your custom connected app, and not the default CLI connected app.

Press Enter to skip sharing the client secret.

6. When you run this command to interact with an agent, specify the username you authorized in the preceding step with the --connected-app-user (-a) flag.

Examples for `agent preview`

Interact with an agent with API name "Resort_Manager" in the org with alias "my-org". Connect to your agent using the alias "my-agent-user"; this alias must point to the username who is authorized using the Web server flow:

```
sf agent preview --api-name "Resort_Manager" --target-org my-org --connected-app-user
my-agent-user
```

Same as the preceding example, but this time save the conversation transcripts to the "./transcripts/my-preview" directory rather than the default "./temp/agent-preview":

```
sf agent preview --api-name "Resort_Manager" --target-org my-org --connected-app-user
my-agent-user --output-dir "transcripts/my-preview"
```

Flags

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-a | --connected-app-user CONNECTED-APP-USER`
Required

Username or alias of the connected app user that's configured with web-based access tokens to the agent.

Type: option

`-n | --api-name API-NAME`
Optional

API name of the agent you want to interact with.

Type: option

`-d | --output-dir OUTPUT-DIR`
Optional

Directory where conversation transcripts are saved.

Type: option

`agent test create` (Beta)

Create an agent test in your org using a local test spec YAML file.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent test create`

To run this command, you must have an agent test spec file, which is a YAML file that lists the test cases for testing a specific agent. Use the "agent generate test-spec" CLI command to generate a test spec file. Then specify the file to this command with the --spec flag, or run this command with no flags to be prompted.

When this command completes, your org contains the new agent test, which you can view and edit using the Testing Center UI. This command also retrieves the metadata component (AiEvaluationDefinition) associated with the new test to your local Salesforce DX project and displays its filename.

After you've created the test in the org, use the "agent test run" command to run it.

Examples for `agent test create`

Create an agent test interactively and be prompted for the test spec and API name of the test in the org; use the default org:

```
sf agent test create
```

Create an agent test and use flags to specify all required information; if a test with same API name already exists in the org, overwrite it without confirmation. Use the org with alias "my-org":

```
sf agent test create --spec specs/Resort_Manager-testSpec.yaml --test-api-name
Resort_Manager_Test --force-overwrite --target-org my-org
```

Preview what the agent test metadata (AiEvaluationDefinition) looks like without deploying it to your default org:

```
sf agent test create --spec specs/Resort_Manager-testSpec.yaml --test-api-name
Resort_Manager_Test --preview
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`--test-api-name TEST-API-NAME`

Optional

API name of the new test; the API name must not exist in the org.

Type: option

--spec SPEC
Optional

Path to the test spec YAML file.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

--preview
Optional

Preview the test metadata file (AiEvaluationDefinition) without deploying to your org.

Type: boolean

--force-overwrite
Optional

Don't prompt for confirmation when overwriting an existing test (based on API name) in your org.

Type: boolean

`agent test list` (Beta)

List the available agent tests in your org.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent test list`

The command outputs a table with the name (API name) of each test along with its unique ID and the date it was created in the org.

Examples for `agent test list`

List the agent tests in your default org:

```
sf agent test list
```

List the agent tests in an org with alias "my-org""

```
sf agent test list --target-org my-org
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

`agent test results` (Beta)

Get the results of a completed agent test run.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent test results`

This command requires a job ID, which the original "agent test run" command displays when it completes. You can also use the --use-most-recent flag to see results for the most recently run agent test.

By default, this command outputs test results in human-readable tables for each test case. The tables show whether the test case passed, the expected and actual values, the test score, how long the test took, and more. Use the --result-format to display the test results in JSON or Junit format. Use the --output-dir flag to write the results to a file rather than to the terminal.

Examples for `agent test results`

Get the results of an agent test run in your default org using its job ID:

```
sf agent test results --job-id 4KBfake0000003F4AQ
```

Get the results of the most recently run agent test in an org with alias "my-org":

```
sf agent test results --use-most-recent --target-org my-org
```

Get the results of the most recently run agent test in your default org, and write the JSON-formatted results into a directory called "test-results":

```
sf agent test results --use-most-recent --output-dir ./test-results --result-format json
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --job-id JOB-ID
Required

Job ID of the completed agent test run.

Type: option

--result-format RESULT-FORMAT
Optional

Format of the agent test run results.

Type: option

Permissible values are: json, human, junit, tap

Default value: human

-d | --output-dir OUTPUT-DIR
Optional

Directory to write the agent test results into.

If the agent test run completes, write the results to the specified directory. If the test is still running, the test results aren't written.

Type: option

`agent test resume` (Beta)

Resume an agent test that you previously started in your org so you can view the test results.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent test resume`

This command requires a job ID, which the original "agent test run" command displays when it completes. You can also use the --use-most-recent flag to see results for the most recently run agent test.

Use the --wait flag to specify the number of minutes for this command to wait for the agent test to complete; if the test completes by the end of the wait time, the command displays the test results. If not, the CLI returns control of the terminal to you, and you must run "agent test resume" again.

By default, this command outputs test results in human-readable tables for each test case. The tables show whether the test case passed, the expected and actual values, the test score, how long the test took, and more. Use the --result-format to display the test results in JSON or Junit format. Use the --output-dir flag to write the results to a file rather than to the terminal.

Examples for `agent test resume`

Resume an agent test in your default org using a job ID:

```
sf agent test resume --job-id 4KBfake0000003F4AQ
```

Resume the most recently-run agent test in an org with alias "my-org" org; wait 10 minutes for the tests to finish:

```
sf agent test resume --use-most-recent --wait 10 --target-org my-org
```

Resume the most recent agent test in your default org, and write the JSON-formatted results into a directory called "test-results":

```
sf agent test resume --use-most-recent --output-dir ./test-results --result-format json
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the original agent test run.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent agent test run.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results to the terminal window.

Type: option

--result-format RESULT-FORMAT
Optional

Format of the agent test run results.

Type: option

Permissible values are: json, human, junit, tap

Default value: human

-d | --output-dir OUTPUT-DIR
Optional

Directory to write the agent test results into.

If the agent test run completes, write the results to the specified directory. If the test is still running, the test results aren't written.

Type: option

`agent test run` (Beta)

Start an agent test in your org.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `agent test run`

Use the --api-name flag to specify the name of the agent test you want to run. Use the output of the "agent test list" command to get the names of all the available agent tests in your org.

By default, this command starts the agent test in your org, but it doesn't wait for the test to finish. Instead, it displays the "agent test resume" command, with a job ID, that you execute to see the results of the test run, and then returns control of the terminal window

to you. Use the --wait flag to specify the number of minutes for the command to wait for the agent test to complete; if the test completes by the end of the wait time, the command displays the test results. If not, run "agent test resume".

By default, this command outputs test results in human-readable tables for each test case, if the test completes in time. The tables show whether the test case passed, the expected and actual values, the test score, how long the test took, and more. Use the --result-format to display the test results in JSON or Junit format. Use the --output-dir flag to write the results to a file rather than to the terminal.

Examples for `agent test run`

Start an agent test called Resort_Manager_Test for an agent in your default org, don't wait for the test to finish:

```
sf agent test run --api-name Resort_Manager_Test
```

Start an agent test for an agent in an org with alias "my-org" and wait for 10 minutes for the test to finish:

```
sf agent test run --api-name Resort_Manager_Test --wait 10 --target-org my-org
```

Start an agent test and write the JSON-formatted results into a directory called "test-results":

```
sf agent test run --api-name Resort_Manager_Test --wait 10 --output-dir ./test-results
--result-format json
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-n | --api-name API-NAME`
Optional

API name of the agent test to run; corresponds to the name of the AiEvaluationDefinition metadata component that implements the agent test.

Type: option

`-w | --wait WAIT`
Optional

Number of minutes to wait for the command to complete and display results to the terminal window.

Type: option

--result-format RESULT-FORMAT

Optional

Format of the agent test run results.

Type: option

Permissible values are: json, human, junit, tap

Default value: human

-d | --output-dir OUTPUT-DIR

Optional

Directory to write the agent test results into.

If the agent test run completes, write the results to the specified directory. If the test is still running, the test results aren't written.

Type: option

`alias` Commands

Use the alias commands to manage your aliases.

alias list

List all aliases currently set on your local computer.

alias set

Set one or more aliases on your local computer.

alias unset

Unset one or more aliases that are currently set on your local computer.

`alias list`

List all aliases currently set on your local computer.

Description for `alias list`

Aliases are global, which means that you can use all the listed aliases in any Salesforce DX project on your computer.

Examples for `alias list`

List all the aliases you've set:

```
sf alias list
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

Aliases for `alias list`

```
force:alias:list
```

`alias set`

Set one or more aliases on your local computer.

Description for `alias set`

Aliases are user-defined short names that make it easier to use the CLI. For example, users often set an alias for a scratch org usernames because they're long and unintuitive. Check the --help of a CLI command to determine where you can use an alias.

You can associate an alias with only one value at a time. If you set an alias multiple times, the alias points to the most recent value. Aliases are global; after you set an alias, you can use it in any Salesforce DX project on your computer.

Use quotes to specify an alias value that contains spaces. You typically use an equal sign to set your alias, although you don't need it if you're setting a single alias in a command.

Examples for `alias set`

Set an alias for a scratch org username:

```
sf alias set my-scratch-org=test-sadbiytjsupn@example.com
```

Set multiple aliases with a single command:

```
sf alias set my-scratch-org=test-sadbiytjsupn@example.com
my-other-scratch-org=test-ss0xut7txzxf@example.com
```

Set an alias that contains spaces:

```
sf alias set my-alias='alias with spaces'
```

Set a single alias without using an equal sign:

```
sf alias set my-scratch-org test-ss0xut7txzxf@example.com
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

Aliases for `alias set`

```
force:alias:set
```

`alias unset`

Unset one or more aliases that are currently set on your local computer.

Description for `alias unset`

Aliases are global, so when you unset one it's no longer available in any Salesforce DX project.

Examples for `alias unset`

Unset an alias:

```
sf alias unset my-alias
```

Unset multiple aliases with a single command:

```
sf alias unset my-alias my-other-alias
```

Unset all aliases:

```
sf alias unset --all [--no-prompt]
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-a | --all
Optional

Unset all currently set aliases.

Type: boolean

-p | --no-prompt
Optional

Don't prompt the user for confirmation when unsetting all aliases.

Type: boolean

Aliases for `alias unset`

```
force:alias:unset
```

`analytics` Commands

Work with analytics assets.

analytics generate template
Generate a simple Analytics template.

`analytics generate template`

Generate a simple Analytics template.

Description for `analytics generate template`

The metadata files associated with the Analytics template must be contained in a parent directory called "waveTemplates" in your package directory. Either run this command from an existing directory of this name, or use the --output-dir flag to generate one or point to an existing one.

Examples for `analytics generate template`

Generate the metadata files for a simple Analytics template file called myTemplate in the force-app/main/default/waveTemplates directory:

```
sf analytics generate template --name myTemplate --output-dir
force-app/main/default/waveTemplates
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-d | --output-dir OUTPUT-DIR`

Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

-n | --name NAME

Required

Name of the Analytics template.

Type: option

Aliases for `analytics generate template`

```
force:analytics:template:create
```

`apex` Commands

Use the apex commands to create Apex classes, execute anonymous blocks, view your logs, run Apex tests, and view Apex test results.

apex generate class
Generate an Apex class.

apex generate trigger
Generate an Apex trigger.

apex get log
Fetch the specified log or given number of most recent logs from the org.

apex get test
Display test results for a specific asynchronous test run.

apex list log
Display a list of IDs and general information about debug logs.

apex run
Execute anonymous Apex code entered on the command line or from a local file.

apex run test
Invoke Apex tests in an org.

apex tail log
Activate debug logging and display logs in the terminal.

`apex generate class`

Generate an Apex class.

Description for `apex generate class`

Generates the Apex *.cls file and associated metadata file. These files must be contained in a parent directory called "classes" in your package directory. Either run this command from an existing directory of this name, or use the --output-dir flag to generate one or point to an existing one.

Examples for `apex generate class`

Generate two metadata files associated with the MyClass Apex class (MyClass.cls and MyClass.cls-meta.xml) in the current directory:

```
sf apex generate class --name MyClass
```

Similar to previous example, but generates the files in the "force-app/main/default/classes" directory:

```
sf apex generate class --name MyClass --output-dir force-app/main/default/classes
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated Apex class.

The name can be up to 40 characters and must start with a letter.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: ApexException, ApexUnitTest, BasicUnitTest, DefaultApexClass, InboundEmailService

Default value: DefaultApexClass

`-d | --output-dir OUTPUT-DIR`
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `apex generate class`

```
force:apex:class:create
```

`apex generate trigger`

Generate an Apex trigger.

Description for `apex generate trigger`

Generates the Apex trigger *.trigger file and associated metadata file. These files must be contained in a parent directory called "triggers" in your package directory. Either run this command from an existing directory of this name, or use the --output-dir flag to generate one or point to an existing one.

If you don't specify the --sobject flag, the .trigger file contains the generic placeholder SOBJECT; replace it with the Salesforce object you want to generate a trigger for. If you don't specify --event, "before insert" is used.

Examples for `apex generate trigger`

Generate two files associated with the MyTrigger Apex trigger (MyTrigger.trigger and MyTrigger.trigger-meta.xml) in the current directory:

```
sf apex generate trigger --name MyTrigger
```

Similar to the previous example, but generate the files in the "force-app/main/default/triggers" directory:

```
sf apex generate trigger --name MyTrigger --output-dir force-app/main/default/triggers
```

Generate files for a trigger that fires on the Account object before and after an insert:

```
sf apex generate trigger --name MyTrigger --sobject Account --event "before insert,after
insert"
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-n | --name NAME

Required

Name of the generated Apex trigger

The name can be up to 40 characters and must start with a letter.

Type: option

-t | --template TEMPLATE
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: ApexTrigger

Default value: ApexTrigger

-d | --output-dir OUTPUT-DIR
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-s | --sobject SOBJECT
Optional

Salesforce object to generate a trigger on.

Type: option

Default value: SOBJECT

-e | --event EVENT
Optional

Events that fire the trigger.

Type: option

Permissible values are: before insert, before update, before delete, after insert, after update, after delete, after undelete

Default value: before insert

Aliases for **apex generate trigger**

```
force:apex:trigger:create
```

apex get log

Fetch the specified log or given number of most recent logs from the org.

Description for `apex get log`

To get the IDs for your debug logs, run "sf apex log list". Executing this command without flags returns the most recent log.

Examples for `apex get log`

Fetch the log in your default org using an ID:

```
sf apex get log --log-id <log id>
```

Fetch the log in the org with the specified username using an ID:

```
sf apex get log --log-id <log id> --target-org me@my.org
```

Fetch the two most recent logs in your default org:

```
sf apex get log --number 2
```

Similar to previous example, but save the two log files in the specified directory:

```
sf apex get log --output-dir /Users/sfdxUser/logs --number 2
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-i | --log-id LOG-ID`
Optional

ID of the specific log to display.

Type: option

`-n | --number NUMBER`
Optional

Number of the most recent logs to display.

Type: option

-d | --output-dir OUTPUT-DIR

Optional

Directory for saving the log files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Aliases for `apex get log`

```
force:apex:log:get
```

`apex get test`

Display test results for a specific asynchronous test run.

Description for `apex get test`

Provide a test run ID to display test results for an enqueued or completed asynchronous test run. The test run ID is displayed after running the "sf apex test run" command.

To see code coverage results, use the --code-coverage flag with --result-format. The output displays a high-level summary of the test run and the code coverage values for classes in your org. If you specify human-readable result format, use the --detailed-coverage flag to see detailed coverage results for each test method run.

Examples for `apex get test`

Display test results for your default org using a test run ID:

```
sf apex get test --test-run-id <test run id>
```

Similar to previous example, but output the result in JUnit format:

```
sf apex get test --test-run-id <test run id> --result-format junit
```

Also retrieve code coverage results and output in JSON format:

```
sf apex get test --test-run-id <test run id> --code-coverage --json
```

Specify a directory in which to save the test results from the org with the specified username (rather than your default org):

```
sf apex get test --test-run-id <test run id> --code-coverage --output-dir <path to outputdir>
 --target-org me@myorg'
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --test-run-id TEST-RUN-ID
Required

ID of the test run.

Type: option

-c | --code-coverage
Optional

Retrieve code coverage results.

Type: boolean

--detailed-coverage
Optional

Display detailed code coverage per test.

Type: boolean

-d | --output-dir OUTPUT-DIR
Optional

Directory in which to store test result files.

Type: option

-r | --result-format RESULT-FORMAT
Optional

Format of the test results.

Type: option

Permissible values are: human, tap, junit, json

Default value: human

--concise
Optional

Display only failed test results; works with human-readable output only.

Type: boolean

Aliases for `apex get test`

```
force:apex:test:report
```

`apex list log`

Display a list of IDs and general information about debug logs.

Description for `apex list log`

Run this command in a project to list the IDs and general information for all debug logs in your default org.

To fetch a specific log from your org, obtain the ID from this command's output, then run the "sf apex log get" command.

Examples for `apex list log`

List the IDs and information about the debug logs in your default org:

```
sf apex list log
```

Similar to previous example, but use the org with the specified username:

```
sf apex list log --target-org me@my.org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `apex list log`

```
force:apex:log:list
```

`apex run`

Execute anonymous Apex code entered on the command line or from a local file.

Description for `apex run`

If you don’t run this command from within a Salesforce DX project, you must specify the —-target-org flag.

To execute your code interactively, run this command with no flags. At the prompt, enter all your Apex code; press CTRL-D when you're finished. Your code is then executed in a single execute anonymous request.

For more information, see "Anonymous Blocks" in the Apex Developer Guide.

Examples for `apex run`

Execute the Apex code that's in the ~/test.apex file in the org with the specified username:

```
sf apex run --target-org testusername@salesforce.org --file ~/test.apex
```

Similar to previous example, but execute the code in your default org:

```
sf apex run --file ~/test.apex
```

Run the command with no flags to start interactive mode; the code will execute in your default org when you exit. At the prompt, start type Apex code and press the Enter key after each line. Press CTRL+D when finished.

```
sf apex run
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-f | --file FILE`
Optional

Path to a local file that contains Apex code.

Type: option

Aliases for `apex run`

```
force:apex:execute
```

`apex run test`

Invoke Apex tests in an org.

Description for `apex run test`

Specify which tests to run by using the --class-names, --suite-names, or --tests flags. Alternatively, use the --test-level flag to run all the tests in your org, local tests, or specified tests.

To see code coverage results, use the --code-coverage flag with --result-format. The output displays a high-level summary of the test run and the code coverage values for classes in your org. If you specify human-readable result format, use the --detailed-coverage flag to see detailed coverage results for each test method run.

By default, Apex tests run asynchronously and immediately return a test run ID. You can use the --wait flag to specify the number of minutes to wait; if the tests finish in that timeframe, the command displays the results. If the tests haven't finished by the end of the wait time, the command displays a test run ID. Use the "sf apex get test --test-run-id" command to get the results.

You must have the "View All Data" system permission to use this command. The permission is disabled by default and can be enabled only by a system administrator.

NOTE: The testRunCoverage value (JSON and JUnit result formats) is a percentage of the covered lines and total lines from all the Apex classes evaluated by the tests in this run.

Examples for `apex run test`

Run all Apex tests and suites in your default org:

```
sf apex run test
```

Run the specified Apex test classes in your default org and display results in human-readable form:

```
sf apex run test --class-names MyClassTest --class-names MyOtherClassTest --result-format
 human
```

Run the specified Apex test suites in your default org and include code coverage results and additional details:

```
sf apex run test --suite-names MySuite --suite-names MyOtherSuite --code-coverage
--detailed-coverage
```

Run the specified Apex tests in your default org and display results in human-readable output:

```
sf apex run test --tests MyClassTest.testCoolFeature --tests MyClassTest.testAwesomeFeature
 --tests AnotherClassTest --tests namespace.TheirClassTest.testThis --result-format human
```

Run all tests in the org with the specified username with the specified test level; save the output to the specified directory:

```
sf apex run test --test-level RunLocalTests --output-dir <path to outputdir> --target-org
 me@my.org
```

Run all tests in the org asynchronously:

```
sf apex run test --target-org myscratch
```

Run all tests synchronously; the command waits to display the test results until all tests finish:

```
sf apex run test --synchronous
```

Run specific tests using the --test-level flag:

```
sf apex run test --test-level RunLocalTests
```

Run Apex tests on all the methods in the specified class; output results in Test Anything Protocol (TAP) format and request code coverage results:

```
sf apex run test --class-names TestA --class-names TestB --result-format tap --code-coverage
```

Run Apex tests on methods specified using the standard Class.method notation; if you specify a test class without a method, the command runs all methods in the class:

```
sf apex run test --tests TestA.excitingMethod --tests TestA.boringMethod --tests TestB
```

Run Apex tests on methods specified using the standard Class.method notation with a namespace:

```
sf apex run test --tests ns.TestA.excitingMethod --tests ns.TestA.boringMethod --tests
ns.TestB
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-c | --code-coverage
Optional

Retrieve code coverage results.

Type: boolean

-d | --output-dir OUTPUT-DIR
Optional

Directory in which to store test run files.

Type: option

-l | --test-level TEST-LEVEL
Optional

Level of tests to run; default is RunLocalTests.

Here's what the levels mean:

- RunSpecifiedTests — Only the tests that you specify in the runTests option are run. Code coverage requirements differ from the default coverage requirements when using this test level. The executed tests must cover each class and trigger in the deployment package for a minimum of 75% code coverage. This coverage is computed for each class and triggers individually, and is different than the overall coverage percentage.

- RunLocalTests — All local tests in your org, including tests that originate from no-namespaced unlocked packages, are run. The tests that originate from installed managed packages and namespaced unlocked packages aren't run. This test level is the default for production deployments that include Apex classes or triggers.

- RunAllTestsInOrg — All tests are run. The tests include all tests in your org.

Type: option

Permissible values are: RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests

-n | --class-names CLASS-NAMES
Optional

Apex test class names to run; default is all classes.

If you select --class-names, you can't specify --suite-names or --tests.

For multiple classes, repeat the flag for each.

--class-names Class1 --class-names Class2

Type: option

-r | --result-format RESULT-FORMAT
Optional

Format of the test results.

Type: option

Permissible values are: human, tap, junit, json

Default value: human

-s | --suite-names SUITE-NAMES
Optional

Apex test suite names to run.

If you select --suite-names, you can't specify --class-names or --tests.

For multiple suites, repeat the flag for each.

--suite-names Suite1 --suite-names Suite2

Type: option

-t | --tests TESTS
Optional

Apex test class names or IDs and, if applicable, test methods to run; default is all tests.

If you specify --tests, you can't specify --class-names or --suite-names

For multiple tests, repeat the flag for each.

--tests Test1 --tests Test2

Type: option

-w | --wait WAIT
Optional

Sets the streaming client socket timeout in minutes; specify a longer wait time if timeouts occur frequently.

Type: option

-y | --synchronous
Optional

Runs test methods from a single Apex class synchronously; if not specified, tests are run asynchronously.

Type: boolean

-v | --detailed-coverage
Optional

Display detailed code coverage per test.

Type: boolean

--concise
Optional

Display only failed test results; works with human-readable output only.

Type: boolean

Aliases for `apex run test`

```
force:apex:test:run
```

`apex tail log`

Activate debug logging and display logs in the terminal.

Description for `apex tail log`

You can also pipe the logs to a file.

Examples for `apex tail log`

Activate debug logging:

```
sf apex tail log
```

Specify a debug level:

```
sf apex tail log --debug-level MyDebugLevel
```

Skip the trace flag setup and apply default colors:

```
sf apex tail log --color --skip-trace-flag
```

Flags

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-c | --color
Optional

Apply default colors to noteworthy log lines.

Type: boolean

-d | --debug-level DEBUG-LEVEL
Optional

Debug level to set on the DEVELOPER_LOG trace flag for your user.

Type: option

-s | --skip-trace-flag
Optional

Skip trace flag setup. Assumes that a trace flag and debug level are fully set up.

Type: boolean

Aliases for **apex tail log**

```
force:apex:log:tail
```

api Commands

Commands to interact with API calls.

`api request graphql` (Beta)

Execute a GraphQL statement.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `api request graphql`

Specify the GraphQL statement with the "--body" flag, either directly at the command line or with a file that contains the statement. You can query Salesforce records using a "query" statement or use mutations to modify Salesforce records.

This command uses the GraphQL API to query or modify Salesforce objects. For details about the API, and examples of queries and mutations, see https://developer.salesforce.com/docs/platform/graphql/guide/graphql-about.html.

Examples for `api request graphql`

Execute a GraphQL query on the Account object by specifying the query directly to the "--body" flag; the command uses your default org:

```
sf api request graphql --body "query accounts { uiapi { query { Account { edges { node {
Id \n Name { value } } } } } } }"
```

Read the GraphQL statement from a file called "example.txt" and execute it on an org with alias "my-org":

```
sf api request graphql --body example.txt --target-org my-org
```

Pipe the GraphQL statement that you want to execute from standard input to the command:

```
$ echo graphql | sf api request graphql --body -
```

Write the output of the command to a file called "output.txt" and include the HTTP response status and headers:

```
sf api request graphql --body example.txt --stream-to-file output.txt --include
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-S | --stream-to-file STREAM-TO-FILE
Optional

Stream responses to a file.

Type: option

-i | --include
Optional

Include the HTTP response status and headers in the output.

Type: boolean

--body BODY
Required

File or content with the GraphQL statement. Specify "-" to read from standard input.

Type: option

`api request rest` (Beta)

Make an authenticated HTTP request using the Salesforce REST API.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `api request rest`

When sending the HTTP request with the "--body" flag, you can specify the request directly at the command line or with a file that contains the request.

For a full list of supported REST endpoints and resources, see https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_list.htm.

Examples for `api request rest`

List information about limits in the org with alias "my-org":

```
sf api request rest 'services/data/v56.0/limits' --target-org my-org
```

List all endpoints in your default org; write the output to a file called "output.txt" and include the HTTP response status and headers:

```
sf api request rest '/services/data/v56.0/' --stream-to-file output.txt --include
```

Get the response in XML format by specifying the "Accept" HTTP header:

```
sf api request rest '/services/data/v56.0/limits' --header 'Accept: application/xml'
```

Create an account record using the POST method; specify the request details directly in the "--body" flag:

```
sf api request rest /services/data/v56.0/sobjects/account --body "{\"Name\" : \"Account
from REST API\",\"ShippingCity\" : \"Boise\"}" --method POST
```

Create an account record using the information in a file called "info.json" (note the @ prefixing the file name):

```
sf api request rest '/services/data/v56.0/sobjects/account' --body @info.json --method
POST
```

Update an account record using the PATCH method:

```
sf api request rest '/services/data/v56.0/sobjects/account/<Account ID>' --body
"{\"BillingCity\": \"San Francisco\"}" --method PATCH
```

Store the values for the request header, body, and so on, in a file, which you then specify with the --file flag; see the description of --file for more information:

```
sf api request rest --file myFile.json
```

Flags

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-i | --include
Optional

Include the HTTP response status and headers in the output.

Type: boolean

-X | --method METHOD
Optional

HTTP method for the request.

Type: option

Permissible values are: GET, POST, PUT, PATCH, HEAD, DELETE, OPTIONS, TRACE

-H | --header HEADER
Optional

HTTP header in "key:value" format.

Type: option

-f | --file FILE

Optional

JSON file that contains values for the request header, body, method, and URL.

Use this flag instead of specifying the request details with individual flags, such as --body or --method. This schema defines how to create the JSON file:

{

url: { raw: string } | string;

method: 'GET', 'POST', 'PUT', 'PATCH', 'HEAD', 'DELETE', 'OPTIONS', 'TRACE';

description?: string;

header: string | Array<Record<string, string>>;

body: { mode: 'raw' | 'formdata'; raw: string; formdata: FormData };

}

Salesforce CLI defined this schema to be mimic Postman schemas; both share similar properties. The CLI's schema also supports Postman Collections to reuse and share requests. As a result, you can build an API call using Postman, export and save it to a file, and then use the file as a value to this flag. For information about Postman, see https://learning.postman.com/.

Here's a simple example of a JSON file that contains values for the request URL, method, and body:

{

"url": "sobjects/Account/<Account ID>",

"method": "PATCH",

"body" : {

"mode": "raw",

"raw": {

"BillingCity": "Boise"

}

}

}

See more examples in the plugin-api test directory, including JSON files that use "formdata" to define collections: https://github.com/salesforcecli/plugin-api/tree/main/test/test-files/data-project.

Type: option

-S | --stream-to-file STREAM-TO-FILE

Optional

Stream responses to a file.

Type: option

-b | --body BODY

Optional

File or content for the body of the HTTP request. Specify "-" to read from standard input or "" for an empty body. If passing a file, prefix the filename with '@'.

Type: option

`cmdt` Commands

Generate custom metadata types and their records.

`cmdt generate field`

Generate a field for a custom metadata type based on the provided field type.

Description for `cmdt generate field`

Similar to a custom object, a custom metadata type has a list of custom fields that represent aspects of the metadata.

This command creates a metadata file that describes the new custom metadata type field. By default, the file is created in a "fields" directory in the current directory. Use the --output-directory to generate the file in the directory that contains the custom metadata type metdata files, such as "force-app/main/default/objects/MyCmdt__mdt" for the custom metadata type called MyCmdt.

Examples for `cmdt generate field`

Generate a metadata file for a custom checkbox field and add the file to the MyCmdt__mdt/fields directory:

```
sf cmdt generate field --name MyCheckboxField --type Checkbox --output-directory
force-app/main/default/objects/MyCmdt__mdt
```

Generate a metadata file for a custom picklist field and add a few values:

```
sf cmdt generate field --name MyPicklistField --type Picklist --picklist-values A
--picklist-values B --picklist-values C --output-directory
force-app/main/default/objects/MyCmdt__mdt
```

Generate a metadata file for a custom number field and specify 2 decimal places:

```
sf cmdt generate field --name MyNumberField --type Number --decimal-places 2
--output-directory force-app/main/default/objects/MyCmdt__mdt
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Unique name for the field.

Type: option

-f | --type TYPE
Required

Type of the field.

You can't use this command to create a custom metadata type field of type "Metadata Relationship". Use the Salesforce Setup UI instead.

Type: option

Permissible values are: Checkbox, Date, DateTime, Email, Number, Percent, Phone, Picklist, Text, TextArea, LongTextArea, Url

-p | --picklist-values PICKLIST-VALUES
Optional

Picklist values; required for picklist fields.

Type: option

-s | --decimal-places DECIMAL-PLACES
Optional

Number of decimal places to use for number or percent fields.

The value must be greater than or equal to zero. Default value is 0.

Type: option

-l | --label LABEL
Optional

Label for the field.

Type: option

-d | --output-directory OUTPUT-DIRECTORY
Optional

Directory to store newly-created field definition files.

New files are automatically created in the "fields" directory. The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Aliases for `cmdt generate field`

```
force:cmdt:field:create
```

```
cmdt:field:create
```

`cmdt generate fromorg`

Generate a custom metadata type and all its records from a Salesforce object.

Description for `cmdt generate fromorg`

Use this command to migrate existing custom objects or custom settings in an org to custom metadata types. If a field of the Salesforce object is of an unsupported type, the field type is automatically converted to text. Run "sf cmdt generate field --help" to see the list of supported cmdt field types, listed in the --type flag summary. Use the --ignore-unsupported to ignore these fields.

This command creates the metadata files that describe the new custom metadata type and its fields in the "force-app/main/default/objects/TypeName__mdt" directory by default, where "TypeName" is the value of the required --dev-name flag. Use --type-output-directory to create them in a different directory.

Examples for `cmdt generate fromorg`

Generate a custom metadata type from a custom object called MySourceObject__c in your default org:

```
sf cmdt generate fromorg --dev-name MyCMDT --sobject MySourceObject__c
```

Generate a custom metadata type from a custom object in an org with alias my-scratch-org; ignore unsupported field types instead of converting them to text:

```
sf cmdt generate fromorg --dev-name MyCMDT --sobject MySourceObject__c --ignore-unsupported
 --target-org my-scratch-org
```

Generate a protected custom metadata type from a custom object:

```
sf cmdt generate fromorg --dev-name MyCMDT --sobject MySourceObject__c --visibility Protected
```

Generate a protected custom metadata type from a custom setting with a specific singular and plural label:

```
sf cmdt generate fromorg --dev-name MyCMDT --label "My CMDT" --plural-label "My CMDTs"
--sobject MySourceSetting__c --visibility Protected
```

Generate a custom metadata type and put the resulting metadata files in the specified directory:

```
sf cmdt generate fromorg --dev-name MyCMDT --sobject MySourceObject__c
--type-output-directory path/to/my/cmdt/directory
```

Generate a custom metadata type and put the resulting record metadata file(s) in the specified directory:

```
sf cmdt generate fromorg --dev-name MyCMDT --sobject MySourceObject__c --records-output-dir
 path/to/my/cmdt/record/directory
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-n | --dev-name DEV-NAME
Required

Name of the custom metadata type.

Type: option

-l | --label LABEL
Optional

Label for the custom metadata type.

Type: option

-p | --plural-label PLURAL-LABEL
Optional

Plural version of the label value; if blank, uses label.

Type: option

-v | --visibility VISIBILITY
Optional

Who can see the custom metadata type.

For more information on what each option means, see this topic in Salesforce Help: https://help.salesforce.com/s/articleView?id=sf.custommetadatatypes_ui_create.htm&type=5.

Type: option

Permissible values are: PackageProtected, Protected, Public

Default value: Public

-s | --sobject SOBJECT
Required

API name of the source Salesforce object used to generate the custom metadata type.

Type: option

-i | --ignore-unsupported
Optional

Ignore unsupported field types.

In this context, "ignore" means that the fields aren't created. The default behavior is to create fields of type text and convert the field values to text.

Type: boolean

-d | --type-output-directory TYPE-OUTPUT-DIRECTORY
Optional

Directory to store newly-created custom metadata type files.

Type: option

Default value: force-app/main/default/objects

-r | --records-output-dir RECORDS-OUTPUT-DIR
Optional

Directory to store newly-created custom metadata record files.

Type: option

Default value: force-app/main/default/customMetadata

Aliases for `cmdt generate fromorg`

```
force:cmdt:generate
```

`cmdt generate object`

Generate a new custom metadata type in the current project.

Description for `cmdt generate object`

This command creates a metadata file that describes the new custom metadata type. By default, the file is created in the MyCustomType__mdt directory in the current directory, where MyCustomType is the value of the required --type-name flag. Use the --output-directory to generate the file in a package directory with other custom metadata types, such as "force-app/main/default/objects".

Examples for `cmdt generate object`

Generate a custom metadata type with developer name 'MyCustomType'; this name is also used as the label:

```
sf cmdt generate object --type-name MyCustomType
```

Generate a protected custom metadata type with a specific label:

```
sf cmdt generate object --type-name MyCustomType --label "Custom Type" --plural-label
"Custom Types" --visibility Protected
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --type-name TYPE-NAME
Required

Unique object name for the custom metadata type.

The name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.

Type: option

-l | --label LABEL
Optional

Label for the custom metadata type.

Type: option

-p | --plural-label PLURAL-LABEL
Optional

Plural version of the label value; if blank, uses label.

Type: option

-v | --visibility VISIBILITY
Optional

Who can see the custom metadata type.

For more information on what each option means, see this topic in Salesforce Help: https://help.salesforce.com/s/articleView?id=sf.custommetadatatypes_ui_create.htm&type=5.

Type: option

Permissible values are: PackageProtected, Protected, Public

Default value: Public

-d | --output-directory OUTPUT-DIRECTORY
Optional

Directory to store the newly-created custom metadata type files

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Aliases for `cmdt generate object`

```
force:cmdt:create
```

```
cmdt:create
```

`cmdt generate record`

Generate a new record for a given custom metadata type in the current project.

Description for `cmdt generate record`

The custom metadata type must already exist in your project. You must specify a name for the new record. Use name=value pairs to specify the values for the fields, such as MyTextField="some text here" or MyNumberField=32.

Examples for `cmdt generate record`

Create a record metadata file for custom metadata type 'MyCMT' with specified values for two custom fields:

```
sf cmdt generate record --type-name MyCMT__mdt --record-name MyRecord My_Custom_Field_1=Foo
 My_Custom_Field_2=Bar
```

Create a protected record metadata file for custom metadata type 'MyCMT' with a specific label and values specified for two custom fields:

```
sf cmdt generate record --type-name MyCMT__mdt --record-name MyRecord --label "My Record"
 --protected true My_Custom_Field_1=Foo My_Custom_Field_2=Bar
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-t | --type-name TYPE-NAME`
Required

API name of the custom metadata type to create a record for; must end in "__mdt".

Type: option

`-n | --record-name RECORD-NAME`
Required

Name of the new record.

Type: option

`-l | --label LABEL`
Optional

Label for the new record.

Type: option

`-p | --protected PROTECTED`
Optional

Protect the record when it's in a managed package.

Protected records can only be accessed by code in the same managed package namespace.

Type: option

Permissible values are: true, false

Default value: false

-i | --input-directory INPUT-DIRECTORY
Optional

Directory from which to get the custom metadata type definition from.

Type: option

Default value: force-app/main/default/objects

-d | --output-directory OUTPUT-DIRECTORY
Optional

Directory to store newly-created custom metadata record files.

Type: option

Default value: force-app/main/default/customMetadata

Aliases for `cmdt generate record`

```
force:cmdt:record:create
```

```
cmdt:record:create
```

`cmdt generate records`

Generate new custom metadata type records from a CSV file.

Description for `cmdt generate records`

The custom metadata type must already exist in your project. By default, the Name column is used to determine the record name; use the --name-column flag to specify a different column.

Examples for `cmdt generate records`

Generate record metadata files from values in a CSV file for the custom metadata type MyCmdt. Use 'Name' as the column that specifies the record name:

```
sf cmdt generate records --csv path/to/my.csv --type-name MyCmdt
```

Generate record metadata files from a CSV file in the directory different from the default, and use 'PrimaryKey' as the column that specifies the record name:

```
sf cmdt generate records --csv path/to/my.csv --type-name MyCmdt --input-directory
path/to/my/cmdt/directory --name-column "PrimaryKey"
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-f | --csv CSV
Required

Pathname of the CSV file.

Type: option

-t | --type-name TYPE-NAME
Required

API name of the custom metadata type to create a record for.

The '__mdt' suffix is appended to the end of the name if it's omitted.

Type: option

-i | --input-directory INPUT-DIRECTORY
Optional

Directory from which to get the custom metadata type definition from.

Type: option

Default value: force-app/main/default/objects

-d | --output-directory OUTPUT-DIRECTORY
Optional

Directory to store newly-created custom metadata record files.

Type: option

Default value: force-app/main/default/customMetadata

-n | --name-column NAME-COLUMN
Optional

Column used to determine the name of the record.

Type: option

Default value: Name

Aliases for **cmdt generate records**

```
force:cmdt:record:insert
```

```
cmdt:record:insert
```

`code-analyzer` Commands

Analyze your code to ensure it adheres to best practices.

`code-analyzer config`

Display the current state of configuration for Code Analyzer.

Description for `code-analyzer config`

Code Analyzer gives you the ability to configure settings that modify Code Analyzer's behavior, to override the tags and severity levels of rules, and to configure the engine specific settings. Use this command to see the current state of this configuration. You can also save this state to a YAML-formatted file that you can modify for your needs.

To apply a custom configuration with Code Analyzer, either keep your custom configuration settings in a `code-analyzer.yml` file located in the current folder from which you are executing commands, or specify the location of your custom configuration file to the Code Analyzer commands with the --config-file flag.

We're continually improving Salesforce Code Analyzer. Tell us what you think! Give feedback at https://sfdc.co/CodeAnalyzerFeedback.

Examples for `code-analyzer config`

Display the current state of the Code Analyzer configuration using the default behavior: display top level configuration, display the engine and rule override settings associated with all the rules; and automatically apply any existing custom configuration settings found in a `code-analyzer.yml` or `code-analyzer.yaml` file in the current folder:

```
sf code-analyzer config
```

This example is identical to the previous one, assuming that `./code-analyzer.yml` exists in your current folder.

```
sf code-analyzer config --config-file ./code-analyzer.yml --rule-selector all
```

Write the current state of configuration to the file `code-analyzer.yml`, including any configuration from an existing `code-analyzer.yml` file. The command preserves all values from the original config, but overwrites any comments:

```
sf code-analyzer config --config-file ./code-analyzer.yml --output-file code-analyzer.yml
```

Display the configuration state for just the recommended rules, instead of all the rules:

```
sf code-analyzer config --rule-selector Recommended
```

Display the configuration state associated with all the rules that are applicable to the files targeted within the folder `./src`:

```
sf code-analyzer config --target ./src
```

Display any relevant configuration settings associated with the rule name 'no-undef' from the 'eslint' engine:

```
sf code-analyzer config --rule-selector eslint:no-undef
```

Load an existing configuration file called `existing-config.yml`, and then write the configuration to a new file called `new-config.yml`, the configuration state that is applicable to all rules that are relevant to the workspace located in the current folder:

```
sf code-analyzer config --config-file ./existing-config.yml --workspace . --output-file
./subfolder-config.yml
```

Flags

-w | --workspace WORKSPACE

Optional

Set of files that make up your workspace.

Use the `--workspace` flag to display only the configuration associated with the rules that apply to the files that make up your workspace. Typically, a workspace is a single project folder that contains all your files. But it can also consist of one or more folders, one or more files, and use glob patterns (wildcards). If you specify this flag multiple times, then your workspace is the sum of the files and folders.

This command uses the types of files in the workspace, such as JavaScript or Typescript, to determine the applicable configuration state. For example, if your workspace contains only JavaScript files, then the command doesn't display configuration state associated with TypeScript rules. The command uses a file's extension to determine what kind of file it is, such as ".ts" for TypeScript.

Some engines can be configured to add additional rules based on what it finds in your workspace. For example, if you set the engines.eslint.auto_discover_eslint_config value of your `code-analyzer.yml` file to true, then supplying your workspace allows the "eslint" engine to examine your files in order to find ESLint configuration files that could potentially add in additional rules.

If you specify `--target` but not `--workspace`, then the current folder '.' is used as your workspace.

Type: option

-t | --target TARGET

Optional

Subset of files within your workspace that you want to target for analysis.

Use the `--target` flag to display the configuration state associated with the rules that apply to only a subset of targeted files within your workspace. You can specify a target as a file, a folder, or a glob pattern. If you specify this flag multiple times, then the full list of targeted files is the sum of the files and folders.

The command uses the type of the targeted files, such as JavaScript or Typescript, to determine which configuration state is applicable. For example, if you target only JavaScript files, then the command doesn't display the configuration state associated with TypeScript rules. The command uses a file's extension to determine what kind of file it is, such as ".ts" for TypeScript.

Each targeted file must live within the workspace specified by the `–-workspace` flag.

If you specify `--workspace` but not `--target`, then all the files within your workspace are targeted.

Type: option

-r | --rule-selector RULE-SELECTOR

Optional

Selection of rules, based on engine name, severity level, rule name, tag, or a combination of criteria separated by colons.

Use the `--rule-selector` flag to display only the configuration associated with the rules based on specific criteria. You can select by engine, such as the rules associated with the "retire-js" or "eslint" engine. Or select by the severity of the rules, such as high or moderate. You can also select rules using tag values or rule names.

You can combine different criteria using colons to further filter the list; the colon works as an intersection. For example, `--rule-selector eslint:Security` reduces the output to only contain the configuration state associated with the rules from the "eslint" engine that have the "Security" tag. To add multiple rule selectors together (a union), specify the `--rule-selector` flag multiple times, such as `--rule-selector eslint:Recommended --rule-selector retire-js:3`.

If you don't specify this flag, then the command uses the "all" rule selector.

Run `sf code-analyzer config --rule-selector Recommended` to display the configuration state associated with just the 'Recommended' rules, instead of all the rules.

Type: option

Default value: all

-c | --config-file CONFIG-FILE
Optional

Path to the existing configuration file used to customize the engines and rules.

Use this flag to apply the customizations from a custom Code Analyzer configuration file to be displayed alongside the current Code Analyzer configuration state.

If you don't specify this flag, then the command looks for and applies a file named `code-analyzer.yml` or `code-analyzer.yaml` in your current folder.

Type: option

-f | --output-file OUTPUT-FILE
Optional

Output file to write the configuration state to. The file is written in YAML format.

If you specify a file within folder, such as `--output-file ./config/code-analyzer.yml`, the folder must already exist, or you get an error. If the file already exists, a prompt asks if you want to overwrite it.

If you don't specify this flag, the command outputs the configuration state to the terminal.

Type: option

code-analyzer rules

List the rules that are available to analyze your code.

Description for code-analyzer rules

You can also view details about the rules, such as the engine it's associated with, tags, and description.

Use this command to determine the exact set of rules to analyze your code. The `code-analyzer run` command has similar flags as this command, so once you've determined the flag values for this command that list the rules you want to run, you specify the same values to the `code-analyzer run` command.

We're continually improving Salesforce Code Analyzer. Tell us what you think! Give feedback at https://sfdc.co/CodeAnalyzerFeedback.

Examples for code-analyzer rules

List rules using the default behavior: include rules from all engines that have a "Recommended" tag; display the rules using concise table format; and automatically apply rule or engine overrides if a `code-analyzer.yml` or `code-analyzer.yaml` file exists in the current folder:

```
sf code-analyzer rules
```

The previous example is equivalent to this example:

```
sf code-analyzer rules --rule-selector Recommended --view table --config-file
./code-analyzer.yml
```

List the recommended rules for the "eslint" engine:

```
sf code-analyzer rules --rule-selector eslint:Recommended
```

List all the rules for the "eslint" engine:

```
sf code-analyzer rules --rule-selector eslint
```

The previous example is equivalent to this example:

```
sf code-analyzer rules --rule-selector eslint:all
```

List the details about all rules for all engines; also write the rules in JSON format to a file called "rules.json" in the "out" folder, which must already exist:

```
sf code-analyzer rules --rule-selector all --output-file ./out/rules.json --view detail
```

Get a more accurate list of the rules that apply specifically to your workspace (all the files in the current folder):

```
sf code-analyzer rules --rule-selector all --workspace .
```

List the recommended rules associated with a workspace that targets all the files in the folder "./other-source" and only the Apex class files (extension .cls) under the folder "./force-app":

```
sf code-analyzer rules --rule-selector Recommended --workspace . --target ./other-source
--target ./force-app/**/*.cls
```

List all the "eslint" engine rules that have a moderate severity (3) and the recommended "retire-js" engine rules with any severity:

```
sf code-analyzer rules --rule-selector eslint:3 --rule-selector retire-js:Recommended
```

Similar to the previous example, but apply the rule overrides and engine settings from the configuration file called `code-analyzer2.yml` in the current folder. If, for example, you changed the severity of an "eslint" rule from moderate (3) to high (2) in the configuration file, then that rule isn't listed:

```
sf code-analyzer rules --rule-selector eslint:3 --rule-selector retire-js:Recommended
--config-file ./code-analyzer2.yml
```

List the details of the "getter-return" rule of the "eslint" engine and the rules named "no-inner-declarations" in any engine:

```
sf code-analyzer rules --rule-selector eslint:getter-return --rule-selector
no-inner-declarations --view detail
```

List the details of the recommended "eslint" engine rules that have the tag "problem" and high severity level (2) that apply when targeting the files within the folder "./force-app":

```
sf code-analyzer rules --rule-selector eslint:Recommended:problem:2 --view detail --target
 ./force-app
```

Flags

-w | --workspace WORKSPACE

Optional

Set of files that make up your workspace.

Use the `--workspace` flag to return a more accurate list of the rules that apply to the files that make up your workspace. Typically, a workspace is a single project folder that contains all your files. But it can also consist of one or more folders, one or more files, and use glob patterns (wildcards). If you specify this flag multiple times, then your workspace is the sum of the files and folders.

The command uses the types of files in the workspace, such as JavaScript or Typescript, to determine which rules to list. For example, if your workspace contains only JavaScript files, the command doesn't list TypeScript rules. The command uses a file's extension to determine what kind of file it is, such as ".ts" for TypeScript.

Some engines may be configured to add additional rules based on what it finds in your workspace. For example, if you set the engines.eslint.auto_discover_eslint_config value of your `code-analyzer.yml` file to true, then supplying your workspace allows the "eslint" engine to examine your files in order to find ESLint configuration files that could potentially add in additional rules.

If you specify `--target` but not `--workspace`, then the current folder '.' is used as your workspace.

Type: option

-t | --target TARGET

Optional

Subset of files within your workspace that you want to target for analysis.

Use the `--target` flag to return a more accurate list of the rules that apply to only a subset of targeted files within your workspace. You can specify a target as a file, a folder, or a glob pattern. If you specify this flag multiple times, then the full list of targeted files is the sum of the files and folders.

The command uses the type of the targeted files, such as JavaScript or Typescript, to determine which rules to list. For example, if you target only JavaScript files, the command doesn't list TypeScript rules. The command uses a file's extension to determine what kind of file it is, such as ".ts" for TypeScript.

Each targeted file must live within the workspace specified by the –-workspace flag.

If you specify `--workspace` but not `--target`, then all the files within your workspace are targeted.

Type: option

-r | --rule-selector RULE-SELECTOR

Optional

Selection of rules, based on engine name, severity level, rule name, tag, or a combination of criteria separated by colons.

Use the `--rule-selector` flag to select the list of rules based on specific criteria. For example, you can select by engine, such as the rules associated with the "retire-js" or "eslint" engine. Or select by the severity of the rules, such as high or moderate. You can also select rules using tag values or rule names. Every rule has a name, which is unique within the scope of an engine. Most rules have tags, although it's not required. An example of a tag is "Recommended".

You can combine different criteria using colons to further filter the list; the colon works as an intersection. For example, `--rule-selector eslint:Security` lists rules associated only with the "eslint" engine that have the Security tag. The flag `--rule-selector eslint:Security:3` flag lists the "eslint" rules that have the Security tag and moderate severity (3). To add multiple rule selectors together (a union), specify the `--rule-selector` flag multiple times, such as `--rule-selector eslint:Recommended --rule-selector retire-js:3`.

Run `sf code-analyzer rules --rule-selector all` to list the possible values for engine name, rule name, tags, and severity levels that you can use with this flag.

Type: option

Default value: Recommended

-c | --config-file CONFIG-FILE

Optional

Path to the configuration file used to customize the engines and rules.

Code Analyzer has an internal default configuration for its rule and engine properties. If you want to override these defaults, you can create a Code Analyzer configuration file.

We recommend that you name your Code Analyzer configuration file `code-analyzer.yml` or `code-analyzer.yaml` and put it at the root of your workspace. You then don't need to use this flag when you run the `code-analyzer rules` command from the root of your workspace, because it automatically looks for either file in the current folder, and if found, applies its rule overrides and engine settings. If you want to name the file something else, or put it in an alternative folder, then you must specify this flag.

To help you get started, use the `code-analyzer config` command to create your first Code Analyzer configuration file. With it, you can change the severity of an existing rule, change a rule's tags, and so on. Then use this flag to specify the file so that the command takes your customizations into account.

Type: option

-f | --output-file OUTPUT-FILE
Optional

Name of the file where the selected rules are written. The file format depends on the extension you specify; currently, only .json is supported for JSON-formatted output.

If you specify a file within folder, such as `--output-file ./out/rules.json`, the folder must already exist, or you get an error. If the file already exists, it's overwritten without prompting.

If you don't specify this flag, the command outputs the rules to only the terminal.

Type: option

-v | --view VIEW
Optional

Format to display the rules in the terminal.

The format `table` is concise and shows minimal output, the format `detail` shows all available information.

If you specify neither `--view` nor `--output-file`, then the default table view is shown. If you specify `--output-file` but not `--view`, only summary information is shown in the terminal.

Type: option

Permissible values are: detail, table

`code-analyzer run`

Analyze your code with a selection of rules to ensure good coding practices.

Description for `code-analyzer run`

You can scan your codebase with the recommended rules. Or use flags to filter the rules based on engines (such as "retire-js" or "eslint"), rule names, tags, and more.

If you want to preview the list of rules before you actually run them, use the `code-analyzer rules` command, which also has the `--config-file`, `--rule-selector`, `--target`, and `--workspace` flags that together define the list of rules to be run.

We're continually improving Salesforce Code Analyzer. Tell us what you think! Give feedback at https://sfdc.co/CodeAnalyzerFeedback.

Examples for `code-analyzer run`

Analyze code using the default behavior: analyze all the files in the current folder (default workspace) using the Recommended rules; display the output in the terminal with the concise table view; and automatically apply rule or engine overrides if a `code-analyzer.yml` or `code-analyzer.yaml` file exists in the current folder:

```
sf code-analyzer run
```

The previous example is equivalent to this example:

```
sf code-analyzer run --rule-selector Recommended --workspace . --target . --view table
--config-file ./code-analyzer.yml
```

Analyze the files using the recommended "eslint" rules and show details of the violations:

```
sf code-analyzer run --rule-selector eslint:Recommended --view detail
```

Analyze the files using all the "eslint" rules:

```
sf code-analyzer run --rule-selector eslint
```

The previous example is equivalent to this example:

```
sf code-analyzer run --rule-selector eslint:all
```

Analyze the files using all rules for all engines:

```
sf code-analyzer run --rule-selector all
```

Analyze files using the recommended "retire-js" rules; target all the files in the folder "./other-source" and only the Apex class files (extension .cls) in the folder "./force-app":

```
sf code-analyzer run --rule-selector retire-js:Recommended --target ./other-source --target
 ./force-app/**/*.cls
```

Specify a custom configuration file and output the results to the "results.csv" file in CSV format; the commands fails if it finds a violation that exceeds the moderate severity level (3):

```
sf code-analyzer run --config-file ./code-analyzer2.yml --output-file results.csv
--severity-threshold 3
```

Analyze the files using all the "eslint" engine rules that have a moderate severity (3) and the recommended "retire-js" engine rules with any severity:

```
sf code-analyzer run --rule-selector eslint:3 --rule-selector retire-js:Recommended
```

Analyze the files using only the "getter-return" rule of the "eslint" engine and any rule named "no-inner-declarations" from any engine:

```
sf code-analyzer run --rule-selector eslint:getter-return --rule-selector
no-inner-declarations
```

Flags

`-w | --workspace WORKSPACE`

Optional

Set of files that make up your workspace.

Typically, a workspace is a single project folder that contains all your files. But it can also consist of one or more folders, one or more files, and use glob patterns (wildcards). If you specify this flag multiple times, then your workspace is the sum of the files and folders.

Some engines often need your entire code base to perform an analysis, even if you want to target only a subset of the files within your workspace , such as with the `--target` flag. For example, the Salesforce Graph Engine might need to compile your entire project in order to properly build a graph so it can perform a data flow analysis on the paths that start in your targeted files.

If you don't specify the `--workspace` flag, then the current folder '.' is used as your workspace.

Type: option

Default value: .

-t | --target TARGET

Optional

Subset of files within your workspace to be targeted for analysis.

You can specify a target as a file, a folder, or a glob pattern.

If you specify this flag multiple times, then the full list of targeted files is the sum of the files and folders.

Each targeted file must live within the workspace that you specified with the `--workspace` flag.

If you don't specify the `--target` flag, then all the files within your workspace (specified by the `--workspace` flag) are targeted for analysis.

Type: option

-r | --rule-selector RULE-SELECTOR

Optional

Selection of rules, based on engine name, severity level, rule name, tag, or a combination of criteria separated by colons.

Use the `--rule-selector` flag to select the list of rules to run based on specific criteria. For example, you can select by engine, such as the rules associated with the "retire-js" or "eslint" engine. Or select by the severity of the rules, such as high or moderate. You can also select rules using tag values or rule names. Every rule has a name, which is unique within the scope of an engine. Most rules have tags, although it's not required. An example of a tag is "Recommended".

You can combine different criteria using colons to further filter the list; the colon works as an intersection. For example, `--rule-selector eslint:Security` runs rules associated only with the "eslint" engine that have the Security tag. The flag `--rule-selector eslint:Security:3` flag runs the "eslint" rules that have the Security tag and moderate severity (3). To add multiple rule selectors together (a union), specify the `--rule-selector` flag multiple times, such as `--rule-selector eslint:Recommended --rule-selector retire-js:3`.

Run `sf code-analyzer rules --rule-selector all` to see the possible values for engine name, rule name, tags, and severity levels that you can use with this flag.

Type: option

Default value: Recommended

-s | --severity-threshold SEVERITY-THRESHOLD

Optional

Severity level of a found violation that must be met or exceeded to cause this command to fail with a non-zero exit code.

You can specify either a number (2) or its equivalent string ("High").

Type: option

-v | --view VIEW

Optional

Format to display the command results in the terminal.

The format `table` is concise and shows minimal output, the format `detail` shows all available information.

If you specify neither `--view` nor `--output-file`, then the default table view is shown. If you specify `--output-file` but not `--view`, only summary information is shown.

Type: option

Permissible values are: detail, table

-f | --output-file OUTPUT-FILE

Optional

Name of the file where the analysis results are written. The file format depends on the extension you specify, such as .csv, .html, .xml, and so on.

If you don't specify this flag, the command outputs the results to only the terminal. Use this flag to print the results to a file; the format of the results depends on the extension you provide. For example, `--output-file results.csv` creates a comma-separated values file. You can specify one of these extensions:

- .csv

- .html or .htm

- .json

- .sarif or .sarif.json

- .xml

To output the results to multiple files, specify this flag multiple times. For example: `--output-file results.json --output-file report.html` creates both a JSON results file and an HTML file.

If you specify a file within a folder, such as `--output-file ./out/results.json`, the folder must already exist, or you get an error. If the file already exists, it's overwritten without prompting.

Type: option

-c | --config-file CONFIG-FILE

Optional

Path to the configuration file used to customize the engines and rules.

Code Analyzer has an internal default configuration for its rule and engine properties. If you want to override these defaults, you can create a Code Analyzer configuration file.

We recommend that you name your Code Analyzer configuration file `code-analyzer.yml` or `code-analyzer.yaml` and put it at the root of your workspace. You then don't need to use this flag when you run the `code-analyzer run` command from the root of your workspace, because it automatically looks for either file in the current folder, and if found, applies its rule overrides and engine settings. If you want to name the file something else, or put it in an alternative folder, then you must specify this flag.

To help you get started, use the `code-analyzer config` command to create your first Code Analyzer configuration file. With it, you can change the severity of an existing rule, change a rule's tags, and so on. Then use this flag to specify the file so that the command takes your customizations into account.

Type: option

`community` Commands

Create and publish an Experience Cloud site.

community publish

Publish an Experience Builder site to make it live.

community create

Create an Experience Cloud site using a template.

Description for community create

Run the "community list template" command to see the templates available in your org. See 'Which Experience Cloud Template Should I Use?' in Salesforce Help for more information about the different template types available. (https://help.salesforce.com/s/articleView?id=sf.siteforce_commtemp_intro.htm&type=5)

When you create a site with the Build Your Own (LWR) template, you must also specify the AuthenticationType value using the format templateParams.AuthenticationType=value, where value is AUTHENTICATED or AUTHENTICATED_WITH_PUBLIC_ACCESS_ENABLED. Name and values are case-sensitive. See 'DigitalExperienceBundle' in the Metadata API Developer Guide for more information. (https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_digitalexperiencebundle.htm)

The site creation process is an async job that generates a jobId. To check the site creation status, query the BackgroundOperation object and enter the jobId as the Id. See 'BackgroundOperation' in the Object Reference for the Salesforce Platform for more information. (https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/sforce_api_objects_backgroundoperation.htm)

If the job doesn't complete within 10 minutes, it times out. You receive an error message and must restart the site creation process. Completed jobs expire after 24 hours and are removed from the database.

When you run this command, it creates the site in preview status, which means that the site isn't yet live. After you finish building your site, you can make it live.

If you have an Experience Builder site, publish the site using the "community publish" command to make it live.

If you have a Salesforce Tabs + Visualforce site, to activate the site and make it live, update the status field of the Network type in Metadata API. (https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_network.htm) Alternatively, in Experience Workspaces, go to Administration | Settings, and click Activate.

For Experience Builder sites, activating the site sends a welcome email to site members.

Examples for community create

Create an Experience Cloud site using template 'Customer Service' and URL path prefix 'customers':

```
sf community create --name 'My Customer Site' --template-name 'Customer Service'
--url-path-prefix customers --description 'My customer site'
```

Create a site using 'Partner Central' template:

```
sf community create --name partnercentral --template-name 'Partner Central' --url-path-prefix
 partners
```

Create a site using the 'Build Your Own (LWR)' template with authentication type of UNAUTHENTICATED:

```
sf community create --name lwrsite --template-name 'Build Your Own (LWR)' --url-path-prefix
 lwrsite templateParams.AuthenticationType=UNAUTHENTICATED
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Name of the site to create.

Type: option

-t | --template-name TEMPLATE-NAME
Required

Template to use to create a site.

An example of a template is Customer Service. Run the "community template list" command to see which templates are available in your org.

Type: option

-p | --url-path-prefix URL-PATH-PREFIX
Optional

URL to append to the domain created when Digital Experiences was enabled for this org.

For example, if your domain name is https://MyDomainName.my.site.com and you create a customer site, enter 'customers' to create the unique URL https://MyDomainName.my.site.com/customers.

Type: option

-d | --description DESCRIPTION
Optional

Description of the site.

The description displays in Digital Experiences - All Sites in Setup and helps with site identification.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `community create`

```
force:community:create
```

`community list template`

Retrieve the list of templates available in your org.

Description for `community list template`

See 'Which Experience Cloud Template Should I Use?' (https://help.salesforce.com/s/articleView?id=sf.siteforce_commtemp_intro.htm&type=5) in Salesforce Help for more information about the different template types available for Experience Cloud.

Examples for `community list template`

Retrieve the template list from an org with alias my-scratch-org:

```
sf community list template --target-org my-scratch-org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `community list template`

```
force:community:template:list
```

`community publish`

Publish an Experience Builder site to make it live.

Description for `community publish`

Each time you publish a site, you update the live site with the most recent updates. When you publish an Experience Builder site for the first time, you make the site's URL live and enable login access for site members.

In addition to publishing, you must activate a site to send a welcome email to all site members. Activation is also required to set up SEO for Experience Builder sites. To activate a site, update the status field of the Network type in Metadata API. (https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_network.htm) Alternatively, in Experience Workspaces, go to Administration | Settings, and click Activate.

An email notification informs you when your changes are live on the published site. The site publish process is an async job that generates a jobId. To check the site publish status manually, query the BackgroundOperation object and enter the jobId as the Id. See 'BackgroundOperation' in the Object Reference for the Salesforce Platform for more information. (https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/sforce_api_objects_backgroundoperation.htm)

If the job doesn't complete within 15 minutes, it times out. You receive an error message and must restart the site publish process. Completed jobs expire after 24 hours and are removed from the database.

Examples for `community publish`

Publish the Experience Builder site with name "My Customer Site':

```
sf community publish --name 'My Customer Site'
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the Experience Builder site to publish.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `community publish`

```
force:community:publish
```

`config` Commands

Commands to configure Salesforce CLI.

config get

Get the value of a configuration variable.

config list

List the configuration variables that you've previously set.

config set

Set one or more configuration variables, such as your default org.

config unset

Unset local or global configuration variables.

`config get`

Get the value of a configuration variable.

Description for `config get`

Run "sf config list" to see the configuration variables you've already set and their level (local or global).

Run "sf config set" to set a configuration variable. For the full list of available configuration variables, see https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_dev_cli_config_values.htm.

Examples for `config get`

Get the value of the "target-org" configuration variable.

```
sf config get target-org
```

Get multiple configuration variables and display whether they're set locally or globally:

```
sf config get target-org api-version --verbose
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--verbose
Optional

Display whether the configuration variables are set locally or globally.

Type: boolean

Aliases for `config get`

```
force:config:get
```

`config list`

List the configuration variables that you've previously set.

Description for `config list`

A config variable can be global or local, depending on whether you used the --global flag when you set it. Local config variables apply only to the current project and override global config variables, which apply to all projects. You can set all config variables as environment variables. Environment variables override their equivalent local and global config variables.

The output of this command takes into account your current context. For example, let's say you run this command from a Salesforce DX project in which you've locally set the "target-org" config variable. The command displays the local value, even if you've also set "target-org" globally. If you haven't set the config variable locally, then the global value is displayed, if set. If you set the SF_TARGET_ORG environment variable, it's displayed as such and overrides any locally or globally set "target-org" config variable.

For the full list of available configuration variables, see https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_dev_cli_config_values.htm.

Examples for `config list`

List the global and local configuration variables that apply to your current context:

```
$ sf config list
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

Aliases for `config list`

```
force:config:list
```

`config set`

Set one or more configuration variables, such as your default org.

Description for `config set`

Use configuration variables to set CLI defaults, such as your default org or the API version you want the CLI to use. For example, if you set the "target-org" configuration variable, you don't need to specify it as a "sf deploy metadata" flag if you're deploying to your default org.

Local configuration variables apply only to your current project. Global variables, specified with the --global flag, apply in any Salesforce DX project.

The resolution order if you've set a flag value in multiple ways is as follows:

1. Flag value specified at the command line.

2. Local (project-level) configuration variable.

3. Global configuration variable.

Run "sf config list" to see the configuration variables you've already set and their level (local or global).

If you're setting a single config variable, you don't need to use an equal sign between the variable and value. But you must use the equal sign if setting multiple config variables.

For the full list of available configuration variables, see https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_dev_cli_config_values.htm.

Examples for `config set`

Set the local target-org configuration variable to an org username:

```
sf config set target-org me@my.org
```

Set the local target-org configuration variable to an alias:

```
sf config set target-org my-scratch-org
```

Set the global target-org and target-dev-hub configuration variables using aliases:

```
sf config set --global target-org=my-scratch-org target-dev-hub=my-dev-hub
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

-g | --global
Optional

Set the configuration variables globally, so they can be used from any Salesforce DX project.

Type: boolean

Aliases for `config set`

```
force:config:set
```

`config unset`

Unset local or global configuration variables.

Description for `config unset`

Local configuration variables apply only to your current project. Global configuration variables apply in any Salesforce DX project.

For the full list of available configuration variables, see https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_dev_cli_config_values.htm.

Examples for `config unset`

Unset the local "target-org" configuration variable:

```
sf config unset target-org
```

Unset multiple configuration variables globally:

```
sf config unset target-org api-version --global
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-g | --global
Optional

Unset the configuration variables globally.

Type: boolean

Aliases for `config unset`

```
force:config:unset
```

`data` Commands

Manage records in your org.

data bulk results
Get the results of a bulk ingest job that you previously ran.

data create file
Upload a local file to an org.

data create record
Create and insert a record into a Salesforce or Tooling API object.

data delete bulk
Bulk delete records from an org using a CSV file. Uses Bulk API 2.0.

data delete record
Deletes a single record from a Salesforce or Tooling API object.

data delete resume
Resume a bulk delete job that you previously started. Uses Bulk API 2.0.

data export bulk
Bulk export records from an org into a file using a SOQL query. Uses Bulk API 2.0.

data export resume
Resume a bulk export job that you previously started. Uses Bulk API 2.0.

data export tree
Export data from an org into one or more JSON files.

data get record
Retrieve and display a single record of a Salesforce or Tooling API object.

data import bulk
Bulk import records into a Salesforce object from a CSV file. Uses Bulk API 2.0.

data import resume
Resume a bulk import job that you previously started. Uses Bulk API 2.0.

data import tree
Import data from one or more JSON files into an org.

data query
Execute a SOQL query.

data resume
View the status of a bulk data load job or batch.

data search
Execute a SOSL text-based search query.

data update bulk

Bulk update records to an org from a CSV file. Uses Bulk API 2.0.

data update record

Updates a single record of a Salesforce or Tooling API object.

data update resume

Resume a bulk update job that you previously started. Uses Bulk API 2.0.

data upsert bulk

Bulk upsert records to an org from a CSV file. Uses Bulk API 2.0.

data upsert resume

Resume a bulk upsert job that you previously started. Uses Bulk API 2.0.

`data bulk results`

Get the results of a bulk ingest job that you previously ran.

Description for `data bulk results`

Use this command to get the complete results after running one of the CLI commands that uses Bulk API 2.0 to ingest (import, update, upsert, or delete) large datasets to your org, such as "data import bulk". The previously-run bulk command must have completed; if it's still processing, run the corresponding resume command first, such as "data import resume." Make note of the job ID of the previous bulk command because you use it to run this command.

You can also use this command to get results from running a bulk ingest job with a different tool, such as Data Loader, as long as you have the job ID. For information on Data Loader, see https://developer.salesforce.com/docs/atlas.en-us.dataLoader.meta/dataLoader/data_loader_intro.htm.

This command first displays the status of the previous bulk job, the operation that was executed in the org (such as insert or hard delete), and the updated Salesforce object. The command then displays how many records were processed in total, and how many were successful or failed. Finally, the output displays the names of the generated CSV-formatted files that contain the specific results for each ingested record. Depending on the success or failure of the bulk command, the results files can include the IDs of inserted records or the specific errors. When possible, if the ingest job failed or was aborted, you also get a CSV file with the unprocessed results.

Examples for `data bulk results`

Get results from a bulk ingest job; use the org with alias "my-scratch":

```
sf data bulk results --job-id 7507i000fake341G --target-org my-scratch
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

-i | --job-id JOB-ID
Required

Job ID of the bulk job.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

data create file

Upload a local file to an org.

Description for data create file

This command always creates a new file in the org; you can't update an existing file. After a successful upload, the command displays the ID of the new ContentDocument record which represents the uploaded file.

By default, the uploaded file isn't attached to a record; in the Salesforce UI the file shows up in the Files tab. You can optionally attach the file to an existing record, such as an account, as long as you know its record ID.

You can also give the file a new name after it's been uploaded; by default its name in the org is the same as the local file name.

Examples for data create file

Upload the local file "resources/astro.png" to your default org:

```
sf data create file --file resources/astro.png
```

Give the file a different filename after it's uploaded to the org with alias "my-scratch":

```
sf data create file --file resources/astro.png --title AstroOnABoat.png --target-org
my-scratch
```

Attach the file to a record in the org:

```
sf data create file --file path/to/astro.png --parent-id a03fakeLoJWPIA3
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-t | --title TITLE
Optional

New title given to the file (ContentDocument) after it's uploaded.

Type: option

-f | --file FILE
Required

Path of file to upload.

Type: option

-i | --parent-id PARENT-ID
Optional

ID of the record to attach the file to.

Type: option

data create record

Create and insert a record into a Salesforce or Tooling API object.

Description for data create record

You must specify a value for all required fields of the object.

When specifying fields, use the format <fieldName>=<value>. Enclose all field-value pairs in one set of double quotation marks, delimited by spaces. Enclose values that contain spaces in single quotes.

This command inserts a record into Salesforce objects by default. Use the --use-tooling-api flag to insert into a Tooling API object.

Examples for data create record

Insert a record into the Account object of your default org; only the required Name field has a value:

```
sf data create record --sobject Account --values "Name=Acme"
```

Insert an Account record with values for two fields, one value contains a space; the command uses the org with alias "my-scratch":

```
sf data create record --sobject Account --values "Name='Universal Containers'
Website=www.example.com" --target-org my-scratch
```

Insert a record into the Tooling API object TraceFlag:

```
sf data create record --use-tooling-api --sobject TraceFlag --values
"DebugLevelId=7dl170000008U36AAE StartDate=2022-12-15T00:26:04.000+0000
ExpirationDate=2022-12-15T00:56:04.000+0000 LogType=CLASS_TRACING
TracedEntityId=01p17000000R6bLAAS"
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce or Tooling API object that you're inserting a record into.

Type: option

-v | --values VALUES
Required

Values for the flags in the form <fieldName>=<value>, separate multiple pairs with spaces.

Type: option

-t | --use-tooling-api
Optional

Use Tooling API so you can insert a record in a Tooling API object.

Type: boolean

Aliases for `data create record`

```
force:data:record:create
```

`data delete bulk`

Bulk delete records from an org using a CSV file. Uses Bulk API 2.0.

Description for `data delete bulk`

The CSV file must have only one column ("Id") and then the list of record IDs you want to delete, one ID per line.

When you execute this command, it starts a job, displays the ID, and then immediately returns control of the terminal to you by default. If you prefer to wait, set the --wait flag to the number of minutes; if it times out, the command outputs the IDs. Use the job ID to check the status of the job with the "sf data delete resume" command.

Examples for `data delete bulk`

Bulk delete Account records from your default org using the list of IDs in the "files/delete.csv" file:

```
sf data delete bulk --sobject Account --file files/delete.csv
```

Bulk delete records from a custom object in an org with alias my-scratch and wait 5 minutes for the command to complete:

```
sf data delete bulk --sobject MyObject__c --file files/delete.csv --wait 5 --target-org
my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-f | --file FILE`
Required

CSV file that contains the IDs of the records to update or delete.

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce object, either standard or custom, that you want to update or delete records from.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

-a | --async
The parameter `--async` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Run the command asynchronously.

Type: boolean

--line-ending LINE-ENDING
Optional

Line ending used in the CSV file. Default value on Windows is `CRLF`; on macOS and Linux it's `LF`.

Type: option

Permissible values are: CRLF, LF

--hard-delete
Optional

Mark the records as immediately eligible for deletion by your org. If you don't specify this flag, the deleted records go into the Recycle Bin.

You must have the "Bulk API Hard Delete" system permission to use this flag. The permission is disabled by default and can be enabled only by a system administrator.

Type: boolean

data delete record

Deletes a single record from a Salesforce or Tooling API object.

Description for data delete record

Specify the record you want to delete with either its ID or with a list of field-value pairs that identify the record. If your list of fields identifies more than one record, the delete fails; the error displays how many records were found.

When specifying field-value pairs, use the format <fieldName>=<value>. Enclose all field-value pairs in one set of double quotation marks, delimited by spaces. Enclose values that contain spaces in single quotes.

This command deletes a record from Salesforce objects by default. Use the --use-tooling-api flag to delete from a Tooling API object.

Examples for `data delete record`

Delete a record from Account with the specified (truncated) ID:

```
sf data delete record --sobject Account --record-id 00180XX
```

Delete a record from Account whose name equals "Acme":

```
sf data delete record --sobject Account --where "Name=Acme"
```

Delete a record from Account identified with two field values, one that contains a space; the command uses the org with alias "my-scratch":

```
sf data delete record --sobject Account --where "Name='Universal Containers' Phone='(123)
 456-7890'" --target-org myscratch
```

Delete a record from the Tooling API object TraceFlag with the specified (truncated) ID:

```
sf data delete record --use-tooling-api --sobject TraceFlag --record-id 7tf8c
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-s | --sobject SOBJECT`
Required

API name of the Salesforce or Tooling API object that you're deleting a record from.

Type: option

`-i | --record-id RECORD-ID`
Optional

ID of the record you're deleting.

Type: option

-w | --where WHERE
Optional

List of <fieldName>=<value> pairs that identify the record you want to delete.

Type: option

-t | --use-tooling-api
Optional

Use Tooling API so you can delete a record from a Tooling API object.

Type: boolean

Aliases for `data delete record`

```
force:data:record:delete
```

`data delete resume`

Resume a bulk delete job that you previously started. Uses Bulk API 2.0.

Description for `data delete resume`

The command uses the job ID returned by the "sf data delete bulk" command or the most recently-run bulk delete job.

Examples for `data delete resume`

Resume a bulk delete job from your default org using an ID:

```
sf data delete resume --job-id 750xx000000005sAAA
```

Resume the most recently run bulk delete job for an org with alias my-scratch:

```
sf data delete resume --use-most-recent --target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org. Not required if the "target-org" configuration variable is already set.

Type: option

-i | --job-id JOB-ID
Optional

ID of the job you want to resume.

Type: option

--use-most-recent
Optional

Use the ID of the most recently-run bulk job.

Type: boolean

Default value: true

--wait WAIT
Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

data export bulk

Bulk export records from an org into a file using a SOQL query. Uses Bulk API 2.0.

Description for data export bulk

You can use this command to export millions of records from an org, either to migrate data or to back it up.

Use a SOQL query to specify the fields of a standard or custom object that you want to export. Specify the SOQL query either at the command line with the --query flag or read it from a file with the --query-file flag; you can't specify both flags. The --output-file flag is required, which means you can only write the records to a file, in either CSV or JSON format.

Bulk exports can take a while, depending on how many records are returned by the SOQL query. If the command times out, or you specified the --async flag, the command displays the job ID. To see the status and get the results of the job, run "sf data export resume" and pass the job ID to the --job-id flag.

IMPORTANT: This command uses Bulk API 2.0, which limits the type of SOQL queries you can run. For example, you can't use aggregate functions such as count(). For the complete list of limitations, see the "SOQL Considerations" section in the "Bulk API 2.0 and Bulk API Developer Guide" (https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/queries.htm).

Examples for data export bulk

Export the Id, Name, and Account.Name fields of the Contact object into a CSV-formatted file; if the export doesn't complete in 10 minutes, the command ends and displays a job ID. Use the org with alias "my-scratch":

```
sf data export bulk --query "SELECT Id, Name, Account.Name FROM Contact" --output-file
export-accounts.csv --wait 10 --target-org my-scratch
```

Similar to previous example, but use the default org, export the records into a JSON-formatted file, and include records that have been soft deleted:

```
sf data export bulk --query "SELECT Id, Name, Account.Name FROM Contact" --output-file
export-accounts.json --result-format json --wait 10 --all-rows
```

Export asynchronously; the command immediately returns a job ID that you then pass to the "sf data export resume" command:

```
sf data export bulk --query "SELECT Id, Name, Account.Name FROM Contact" --output-file
export-accounts.json --result-format json --async
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-w | --wait WAIT
Optional

Time to wait for the command to finish, in minutes.

Type: option

--async
The parameter `--async` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Don't wait for the job to complete.

Type: boolean

-q | --query QUERY
Optional

SOQL query to execute.

Type: option

--query-file QUERY-FILE
Optional

File that contains the SOQL query.

Type: option

--all-rows
Optional

Include records that have been soft-deleted due to a merge or delete. By default, deleted records are not returned.

Type: boolean

--output-file OUTPUT-FILE
Required

File where records are written.

Type: option

-r | --result-format RESULT-FORMAT
Required

Format to write the results.

Type: option

Permissible values are: csv, json

Default value: csv

--column-delimiter COLUMN-DELIMITER
Optional

Column delimiter to be used when writing CSV output. Default is COMMA.

Type: option

Permissible values are: BACKQUOTE, CARET, COMMA, PIPE, SEMICOLON, TAB

--line-ending LINE-ENDING
Optional

Line ending to be used when writing CSV output. Default value on Windows is is `CRLF`; on macOS and Linux it's `LR`.

Type: option

Permissible values are: LF, CRLF

data export resume

Resume a bulk export job that you previously started. Uses Bulk API 2.0.

Description for data export resume

When the original "data export bulk" command either times out or is run with the --async flag, it displays a job ID. To see the status and get the results of the bulk export, run this command by either passing it the job ID or using the --use-most-recent flag to specify the most recent bulk export job.

Examples for `data export resume`

Resume a bulk export job run on your default org by specifying a job ID:

```
sf data export resume --job-id 750xx000000005sAAA
```

Resume the most recently-run bulk export job for an org with alias my-scratch:

```
sf data export resume --use-most-recent --target-org my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-i | --job-id JOB-ID`
Optional

Job ID of the bulk export.

Type: option

`--use-most-recent`
Optional

Use the job ID of the bulk export job that was most recently run.

Type: boolean

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`data export tree`

Export data from an org into one or more JSON files.

Description for `data export tree`

Specify a SOQL query, either directly at the command line or read from a file, to retrieve the data you want to export. The exported data is written to JSON files in sObject tree format, which is a collection of nested, parent-child records with a single root record. Use these JSON files to import data into an org with the "sf data import tree" command.

If your SOQL query references multiple objects, the command generates a single JSON file by default. You can specify the --plan flag to generate separate JSON files for each object and a plan definition file that aggregates them. You then specify just this plan definition file when you import the data into an org.

The SOQL query can return a maximum of 2,000 records. For more information, see the REST API Developer Guide. (https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_composite_sobject_tree.htm).

Examples for `data export tree`

Export records retrieved with the specified SOQL query into a single JSON file in the current directory; the command uses your default org:

```
sf data export tree --query "SELECT Id, Name, (SELECT Name, Address__c FROM Properties__r)
 FROM Broker__c"
```

Export data using a SOQL query in the "query.txt" file and generate JSON files for each object and a plan that aggregates them:

```
sf data export tree --query query.txt --plan
```

Prepend "export-demo" before each generated file and generate the files in the "export-out" directory; run the command on the org with alias "my-scratch":

```
sf data export tree --query query.txt --plan --prefix export-demo --output-dir export-out
 --target-org my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-q | --query QUERY`
Required

SOQL query, or filepath of a file that contains the query, to retrieve records.

Type: option

`-p | --plan`
Optional

Generate multiple sObject tree files and a plan definition file for aggregated import.

Type: boolean

-x | --prefix PREFIX
Optional

Prefix of generated files.

Type: option

-d | --output-dir OUTPUT-DIR
Optional

Directory in which to generate the JSON files; default is current directory.

Type: option

Aliases for `data export tree`

```
force:data:tree:export
```

`data get record`

Retrieve and display a single record of a Salesforce or Tooling API object.

Description for `data get record`

Specify the record you want to retrieve with either its ID or with a list of field-value pairs that identify the record. If your list of fields identifies more than one record, the command fails; the error displays how many records were found.

When specifying field-value pairs, use the format <fieldName>=<value>. Enclose all field-value pairs in one set of double quotation marks, delimited by spaces. Enclose values that contain spaces in single quotes.

The command displays all the record's fields and their values, one field per terminal line. Fields with no values are displayed as "null".

This command retrieves a record from Salesforce objects by default. Use the --use-tooling-api flag to retrieve from a Tooling API object.

Examples for `data get record`

Retrieve and display a record from Account with the specified (truncated) ID:

```
sf data get record --sobject Account --record-id 00180XX
```

Retrieve a record from Account whose name equals "Acme":

```
sf data get record --sobject Account --where "Name=Acme"
```

Retrieve a record from Account identified with two field values, one that contains a space; the command uses the org with alias "my-scratch":

```
sf data get record --sobject Account --where "Name='Universal Containers' Phone='(123)
456-7890'" --target-org myscratch
```

Retrieve a record from the Tooling API object TraceFlag with the specified (truncated) ID:

```
sf data get record --use-tooling-api --sobject TraceFlag --record-id 7tf8c
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce or Tooling API object that you're retrieving a record from.

Type: option

-i | --record-id RECORD-ID
Optional

ID of the record you're retrieving.

Type: option

-w | --where WHERE
Optional

List of <fieldName>=<value> pairs that identify the record you want to display.

Type: option

-t | --use-tooling-api
Optional

Use Tooling API so you can retrieve a record from a Tooling API object.

Type: boolean

Aliases for `data get record`

```
force:data:record:get
```

`data import bulk`

Bulk import records into a Salesforce object from a CSV file. Uses Bulk API 2.0.

Description for `data import bulk`

You can use this command to import millions of records into the object from a file in comma-separated values (CSV) format.

All the records in the CSV file must be for the same Salesforce object. Specify the object with the `--sobject` flag.

Bulk imports can take a while, depending on how many records are in the CSV file. If the command times out, or you specified the --async flag, the command displays the job ID. To see the status and get the results of the job, run "sf data import resume" and pass the job ID to the --job-id flag.

For information and examples about how to prepare your CSV files, see "Prepare Data to Ingest" in the "Bulk API 2.0 and Bulk API Developer Guide" (https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/datafiles_prepare_data.htm).

Examples for `data import bulk`

Import Account records from a CSV-formatted file into an org with alias "my-scratch"; if the import doesn't complete in 10 minutes, the command ends and displays a job ID:

```
sf data import bulk --file accounts.csv --sobject Account --wait 10 --target-org my-scratch
```

Import asynchronously and use the default org; the command immediately returns a job ID that you then pass to the "sf data import resume" command:

```
sf data import bulk --file accounts.csv --sobject Account --async
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-a | --async`
The parameter `--async` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Don't wait for the command to complete.

Type: boolean

`-f | --file FILE`
Required

CSV file that contains the Salesforce object records you want to import.

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce object, either standard or custom, into which you're importing records.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-w | --wait WAIT
Optional

Time to wait for the command to finish, in minutes.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--line-ending LINE-ENDING
Optional

Line ending used in the CSV file. Default value on Windows is `CRLF`; on macOS and Linux it's `LF`.

Type: option

Permissible values are: CRLF, LF

--column-delimiter COLUMN-DELIMITER
Optional

Column delimiter used in the CSV file.

Type: option

Permissible values are: BACKQUOTE, CARET, COMMA, PIPE, SEMICOLON, TAB

data import resume

Resume a bulk import job that you previously started. Uses Bulk API 2.0.

Description for data import resume

When the original "sf data import bulk" command either times out or is run with the --async flag, it displays a job ID. To see the status and get the results of the bulk import, run this command by either passing it the job ID or using the --use-most-recent flag to specify the most recent bulk import job.

Examples for data import resume

Resume a bulk import job to your default org using an ID:

```
sf data import resume --job-id 750xx000000005sAAA
```

Resume the most recently run bulk import job for an org with alias my-scratch:

```
sf data import resume --use-most-recent --target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--use-most-recent
Optional

Use the job ID of the bulk import job that was most recently run.

Type: boolean

-i | --job-id JOB-ID
Optional

Job ID of the bulk import.

Type: option

-w | --wait WAIT
Optional

Time to wait for the command to finish, in minutes.

Type: option

data import tree

Import data from one or more JSON files into an org.

Description for data import tree

The JSON files that contain the data are in sObject tree format, which is a collection of nested, parent-child records with a single root record. Use the "sf data export tree" command to generate these JSON files.

If you used the --plan flag when exporting the data to generate a plan definition file, use the --plan flag to reference the file when you import. If you're not using a plan, use the --files flag to list the files. If you specify multiple JSON files that depend on each other in a parent-child relationship, be sure you list them in the correct order.

Examples for data import tree

Import the records contained in two JSON files into the org with alias "my-scratch":

```
sf data import tree --files Contact.json,Account.json --target-org my-scratch
```

Import records using a plan definition file into your default org:

```
sf data import tree --plan Account-Contact-plan.json
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-f | --files FILES
Optional

Comma-separated and in-order JSON files that contain the records, in sObject tree format, that you want to insert.

Type: option

-p | --plan PLAN
Optional

Plan definition file to insert multiple data files.

Unlike when you use the `--files` flag, the files listed in the plan definition file **can** contain more then 200 records. When the CLI executes the import, it automatically batches the records to comply with the 200 record limit set by the API.

The order in which you list the files in the plan definition file matters. Specifically, records with lookups to records in another file should be listed AFTER that file. For example, let's say you're loading Account and Contact records, and the contacts have references to those accounts. Be sure you list the Accounts file before the Contacts file.

The plan definition file has the following schema:

- items(object) - SObject Type: Definition of records to be insert per SObject Type

- sobject(string) - Name of SObject: Child file references must have SObject roots of this type

- files(array) - Files: An array of files paths to load

Type: option

Aliases for `data import tree`

```
force:data:tree:import
```

`data query`

Execute a SOQL query.

Description for `data query`

Specify the SOQL query at the command line with the --query flag or read the query from a file with the --file flag.

If your query returns more than 10,000 records, prefer to use the `sf data export bulk` command instead. It runs the query using Bulk API 2.0, which has higher limits than the default API used by the command.

Examples for `data query`

Specify a SOQL query at the command line; the command uses your default org:

```
sf data query --query "SELECT Id, Name, Account.Name FROM Contact"
```

Read the SOQL query from a file called "query.txt" and write the CSV-formatted output to a file; the command uses the org with alias "my-scratch":

```
sf data query --file query.txt --output-file output.csv --result-format csv --target-org
my-scratch
```

Use Tooling API to run a query on the ApexTrigger Tooling API object:

```
sf data query --query "SELECT Name FROM ApexTrigger" --use-tooling-api
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

-q | --query QUERY
Optional

SOQL query to execute.

Type: option

-f | --file FILE
Optional

File that contains the SOQL query.

Type: option

-t | --use-tooling-api
Optional

Use Tooling API so you can run queries on Tooling API objects.

Type: boolean

--all-rows
Optional

Include deleted records. By default, deleted records are not returned.

Type: boolean

-r | --result-format RESULT-FORMAT
Optional

Format to display the results; the --json flag overrides this flag.

Type: option

Permissible values are: human, csv, json

Default value: human

--output-file OUTPUT-FILE
Optional

File where records are written; only CSV and JSON output formats are supported.

Type: option

Aliases for `data query`

```
force:data:soql:query
```

`data resume`

View the status of a bulk data load job or batch.

Description for `data resume`

Run this command using the job ID or batch ID returned from the "sf data delete bulk" or "sf data upsert bulk" commands.

Examples for `data resume`

View the status of a bulk load job:

```
sf data resume --job-id 750xx000000005sAAA
```

View the status of a bulk load job and a specific batches:

```
sf data resume --job-id 750xx000000005sAAA --batch-id 751xx000000005nAAA
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-b | --batch-id BATCH-ID`
Optional

ID of the batch whose status you want to view; you must also specify the job ID.

Type: option

`-i | --job-id JOB-ID`
Required

ID of the job whose status you want to view.

Type: option

`data search`

Execute a SOSL text-based search query.

Description for `data search`

Specify the SOSL query at the command line with the --query flag or read the query from a file with the --file flag.

By default, the results are written to the terminal in human-readable format. If you specify `--result-format csv`, the output is written to one or more CSV (comma-separated values) files. The file names correspond to the Salesforce objects in the results, such as Account.csv. Both `--result-format human` and `--result-format json` display only to the terminal.

Examples for `data search`

Specify a SOSL query at the command line; the command uses your default org:

```
sf data search --query "FIND {Anna Jones} IN Name Fields RETURNING Contact (Name, Phone)"
```

Read the SOSL query from a file called "query.txt"; the command uses the org with alias "my-scratch":

```
sf data search --file query.txt --target-org my-scratch
```

Similar to the previous example, but write the results to one or more CSV files, depending on the Salesforce objects in the results:

```
sf data search --file query.txt --target-org my-scratch --result-format csv
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-q | --query QUERY`
Optional

SOSL query to execute.

Type: option

`-f | --file FILE`
Optional

File that contains the SOSL query.

Type: option

-r | --result-format RESULT-FORMAT

Optional

Format to display the results, or to write to disk if you specify "csv".

Type: option

Permissible values are: human, csv, json

Default value: human

data update bulk

Bulk update records to an org from a CSV file. Uses Bulk API 2.0.

Description for data update bulk

You can use this command to update millions of Salesforce object records based on a file in comma-separated values (CSV) format.

All the records in the CSV file must be for the same Salesforce object. Specify the object with the `--sobject` flag. The first column of every line in the CSV file must be an ID of the record you want to update. The CSV file can contain only existing records; if a record in the file doesn't currently exist in the Salesforce object, the command fails. Consider using "sf data upsert bulk" if you also want to insert new records.

Bulk updates can take a while, depending on how many records are in the CSV file. If the command times out, or you specified the --async flag, the command displays the job ID. To see the status and get the results of the job, run "sf data update resume" and pass the job ID to the --job-id flag.

For information and examples about how to prepare your CSV files, see "Prepare Data to Ingest" in the "Bulk API 2.0 and Bulk API Developer Guide" (https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/datafiles_prepare_data.htm).

Examples for data update bulk

Update Account records from a CSV-formatted file into an org with alias "my-scratch"; if the update doesn't complete in 10 minutes, the command ends and displays a job ID:

```
sf data update bulk --file accounts.csv --sobject Account --wait 10 --target-org my-scratch
```

Update asynchronously and use the default org; the command immediately returns a job ID that you then pass to the "sf data update resume" command:

```
sf data update bulk --file accounts.csv --sobject Account --async
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-a | --async
The parameter `--async` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Don't wait for the command to complete.

Type: boolean

-w | --wait WAIT
Optional

Time to wait for the command to finish, in minutes.

Type: option

-f | --file FILE
Required

CSV file that contains the Salesforce object records you want to update.

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce object, either standard or custom, which you are updating.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--line-ending LINE-ENDING
Optional

Line ending used in the CSV file. Default value on Windows is `CRLF`; on macOS and Linux it's `LF`.

Type: option

Permissible values are: CRLF, LF

--column-delimiter COLUMN-DELIMITER
Optional

Column delimiter used in the CSV file.

Type: option

Permissible values are: BACKQUOTE, CARET, COMMA, PIPE, SEMICOLON, TAB

data update record

Updates a single record of a Salesforce or Tooling API object.

Description for `data update record`

Specify the record you want to update with either its ID or with a list of field-value pairs that identify the record. If your list of fields identifies more than one record, the update fails; the error displays how many records were found.

When using field-value pairs for both identifying the record and specifiyng the new field values, use the format <fieldName>=<value>. Enclose all field-value pairs in one set of double quotation marks, delimited by spaces. Enclose values that contain spaces in single quotes.

This command updates a record in Salesforce objects by default. Use the --use-tooling-api flag to update a Tooling API object.

Examples for `data update record`

Update the Name field of an Account record with the specified (truncated) ID:

```
sf data update record --sobject Account --record-id 001D0 --values "Name=NewAcme"
```

Update the Name field of an Account record whose current name is 'Old Acme':

```
sf data update record --sobject Account --where "Name='Old Acme'" --values "Name='New
Acme'"
```

Update the Name and Website fields of an Account record with the specified (truncated) ID:

```
sf data update record --sobject Account --record-id 001D0 --values "Name='Acme III'
Website=www.example.com"
```

Update the ExpirationDate field of a record of the Tooling API object TraceFlag using the specified (truncated) ID:

```
sf data update record -t --sobject TraceFlag --record-id 7tf170000009cUBAAY --values
"ExpirationDate=2017-12-01T00:58:04.000+0000"
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce or Tooling API object that contains the record you're updating.

Type: option

-i | --record-id RECORD-ID
Optional

ID of the record you're updating.

Type: option

-w | --where WHERE
Optional

List of <fieldName>=<value> pairs that identify the record you want to update.

Type: option

-v | --values VALUES
Required

Fields that you're updating, in the format of <fieldName>=<value> pairs.

Type: option

-t | --use-tooling-api
Optional

Use Tooling API so you can update a record in a Tooling API object.

Type: boolean

Aliases for `data update record`

```
force:data:record:update
```

`data update resume`

Resume a bulk update job that you previously started. Uses Bulk API 2.0.

Description for `data update resume`

When the original "sf data update bulk" command either times out or is run with the --async flag, it displays a job ID. To see the status and get the results of the bulk update, run this command by either passing it the job ID or using the --use-most-recent flag to specify the most recent bulk update job.

Examples for `data update resume`

Resume a bulk update job of your default org using a job ID:

```
sf data update resume --job-id 750xx000000005sAAA
```

Resume the most recently run bulk update job for an org with alias "my-scratch":

```
sf data update resume --use-most-recent --target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--use-most-recent
Optional

Use the job ID of the bulk update job that was most recently run.

Type: boolean

-i | --job-id JOB-ID
Optional

Job ID of the bulk update.

Type: option

-w | --wait WAIT
Optional

Time to wait for the command to finish, in minutes.

Type: option

data upsert bulk

Bulk upsert records to an org from a CSV file. Uses Bulk API 2.0.

Description for data upsert bulk

An upsert refers to inserting a record into a Salesforce object if the record doesn't already exist, or updating it if it does exist.

When you execute this command, it starts a job, displays the ID, and then immediately returns control of the terminal to you by default. If you prefer to wait, set the --wait flag to the number of minutes; if it times out, the command outputs the IDs. Use the job and batch IDs to check the status of the job with the "sf data upsert resume" command.

See "Prepare CSV Files" in the Bulk API Developer Guide for details on formatting your CSV file. (https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/datafiles_prepare_csv.htm)

Examples for data upsert bulk

Bulk upsert records to the Contact object in your default org:

```
sf data upsert bulk --sobject Contact --file files/contacts.csv --external-id Id
```

Bulk upsert records to a custom object in an org with alias my-scratch and wait 5 minutes for the command to complete:

```
sf data upsert bulk --sobject MyObject__c --file files/file.csv --external-id MyField__c
--wait 5 --target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-f | --file FILE
Required

CSV file that contains the IDs of the records to update or delete.

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce object, either standard or custom, that you want to update or delete records from.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

-a | --async
The parameter `--async` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Run the command asynchronously.

Type: boolean

--line-ending LINE-ENDING
Optional

Line ending used in the CSV file. Default value on Windows is `CRLF`; on macOS and Linux it's `LF`.

Type: option

Permissible values are: CRLF, LF

--column-delimiter COLUMN-DELIMITER
Optional

Column delimiter used in the CSV file.

Type: option

Permissible values are: BACKQUOTE, CARET, COMMA, PIPE, SEMICOLON, TAB

-i | --external-id EXTERNAL-ID
Required

Name of the external ID field, or the Id field.

Type: option

data upsert resume

Resume a bulk upsert job that you previously started. Uses Bulk API 2.0.

Description for data upsert resume

The command uses the job ID returned from the "sf data upsert bulk" command or the most recently-run bulk upsert job.

Examples for data upsert resume

Resume a bulk upsert job from your default org using an ID:

```
sf data upsert resume --job-id 750xx000000005sAAA
```

Resume the most recently run bulk upsert job for an org with alias my-scratch:

```
sf data upsert resume --use-most-recent --target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org. Not required if the "target-org" configuration variable is already set.

Type: option

-i | --job-id JOB-ID

Optional

ID of the job you want to resume.

Type: option

--use-most-recent

Optional

Use the ID of the most recently-run bulk job.

Type: boolean

Default value: true

--wait WAIT

Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

dev Commands

Commands for sf plugin development.

dev audit messages

Audit messages in a plugin's messages directory to locate unused messages and missing messages that have references in source code.

Examples for `dev audit messages`

Audit messages using default directories:

```
sf dev audit messages
```

Audit messages in the "messages" directory in the current working directory; the plugin's source directory is in "src":

```
sf dev audit messages --messages-dir ./messages --source-dir ./src
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-p | --project-dir PROJECT-DIR`
Optional

Location of the project where messages are to be audited.

Type: option

Default value: .

`-m | --messages-dir MESSAGES-DIR`
Optional

Directory that contains the plugin's message files.

The default is the "messages" directory in the current working directory.

Type: option

Default value: messages

`-s | --source-dir SOURCE-DIR`
Optional

Directory that contains the plugin's source code.

The default is the "src" directory in the current working directory.

Type: option

Default value: src

`dev convert messages`

Convert a .json messages file into Markdown.

Description for `dev convert messages`

Preserves the filename and the original messages file, then creates a new file with the Markdown extension and standard headers for the command and flag summaries, descriptions, and so on. After you review the new Markdown file, delete the old .json file.

Examples for `dev convert messages`

Convert the my-command.json message file into my-command.md with the standard messages headers:

```
sf dev convert messages --filename my-command.json
```

Similar to previous example, but specify the plugin project directory:

```
sf dev convert messages --project-dir ./path/to/plugin --filename my-command.json
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-p | --project-dir PROJECT-DIR`
Optional

Location of the project whose messages are to be converted.

Type: option

Default value: .

`-f | --file-name FILE-NAME`
Required

Filename to convert.

Type: option

`dev convert script`

Convert a script file that contains deprecated sfdx-style commands to use the new sf-style commands instead.

Description for `dev convert script`

Important: Use this command only to get started on the sfdx->sf script migration. We don't guarantee that the new sf-style command replacements work correctly or as you expect. You must test, and probably update, the new script before putting it into production. We also don't guarantee that the JSON results are the same as before.

This command can convert a large part of your script, but possibly not all. There are some sfdx-style commands that don't have an obvious sf-style equivalent. In this case, this command doesn't replace the sfdx-style command but instead adds a comment to remind

you that you must convert it manually. See the Salesforce CLI Command Reference for migration information about each deprecated sfdx-style command: https://developer.salesforce.com/docs/atlas.en-us.sfdx_cli_reference.meta/sfdx_cli_reference/cli_reference.htm.

This command is interactive; as it scans your script, it prompts you when it finds an sfdx-style command or flag and asks if you want to convert it to the displayed suggestion. The command doesn't update the script file directly; rather, it creates a new file whose name is the original name but with "-converted" appended to it. The script replaces all instances of "sfdx" with "sf". For each prompt you answer "y" to, the command replaces the sfdx-style names with their equivalent sf-style ones. For example, "sfdx force:apex:execute --targetusername myscratch" is replaced with "sf apex run --target-org myscratch".

Examples for `dev convert script`

Convert the YAML file called "myScript.yml" located in the current directory; the new file that contains the replacements is called "myScript-converted.yml":

```
sf dev convert script --script ./myScript.yml
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-s | --script SCRIPT`
Required

Filepath to the script you want to convert.

Type: option

`dev generate command`

Generate a new sf command.

Description for `dev generate command`

You must run this command from within a plugin directory, such as the directory created with the "sf dev generate plugin" command.

The command generates basic source files, messages (*.md), and test files for your new command. The Typescript files contain import statements for the minimum required Salesforce libraries, and scaffold some basic code. The new type names come from the value you passed to the --name flag.

The command updates the package.json file, so if it detects conflicts with the existing file, you're prompted whether you want to overwrite the file. There are a number of package.json updates required for a new command, so we recommend you answer "y" so the command takes care of them all. If you answer "n", you must update the package.json file manually.

Examples for `dev generate command`

Generate the files for a new "sf my exciting command":

```
sf dev generate command --name my:exciting:command
```

Flags

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the new command. Use colons to separate the topic and command names.

Type: option

`--force`
Optional

Overwrite existing files.

Type: boolean

`--dry-run`
Optional

Display the changes that would be made without writing them to disk.

Type: boolean

`--nuts`
Optional

Generate a NUT test file for the command.

Type: boolean

Default value: true

`--unit`
Optional

Generate a unit test file for the command.

Type: boolean

Default value: true

`dev generate flag`

Generate a flag for an existing command.

Description for `dev generate flag`

You must run this command from within a plugin directory, such as the directory created with the "sf dev generate plugin" command.

This command is interactive. It first discovers all the commands currently implemented in the plugin, and asks you which you want to create a new flag for. It then prompts for other flag properties, such as its long name, optional short name, type, whether it's required, and so on. Long flag names must be kebab-case and not camelCase. The command doesn't let you use an existing long or short flag name. When the command completes, the Typescript file for the command is updated with the code for the new flag.

Use the --dry-run flag to review new code for the command file without actually updating it.

Examples for `dev generate flag`

Generate a new flag and update the command file:

```
sf dev generate flag
```

Don't actually update the command file, just view the generated code:

```
sf dev generate flag --dry-run
```

Flags

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-d | --dry-run`

Optional

Print new flag code instead of adding it to the command file.

Type: boolean

`dev generate plugin`

Generate a new sf plugin.

Description for `dev generate plugin`

This command is interactive. You're prompted for information to populate your new plugin, such as its name, description, author, and percentage of code coverage you want. The command clones the 'salesforcecli/plugin-template-sf' GitHub repository, installs the plug-in's npm package dependencies using yarn install, and updates the package properties.

When the command completes, your new plugin contains the source, message, and test files for a sample "sf hello world" command.

Examples for `dev generate plugin`

```
sf dev generate plugin
```

Flags

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`--dry-run`

Optional

Display the changes that would be made without writing them to disk.

Type: boolean

Aliases for `dev generate plugin`

```
plugins:generate
```

`doctor` Commands

Tools for diagnosing problems with Salesforce CLI.

doctor

Gather CLI configuration data and run diagnostic tests to discover and report potential problems in your environment.

`doctor`

Gather CLI configuration data and run diagnostic tests to discover and report potential problems in your environment.

Description for `doctor`

When you run the doctor command without parameters, it first displays a diagnostic overview of your environment. It then writes a detailed diagnosis to a JSON file in the current directory. Use the --outputdir to specify a different directory. To run diagnostic tests on a specific plugin, use the --plugin parameter. If the plugin isn't listening to the doctor, then you get a warning.

Use the --command parameter to run a specific command in debug mode; the doctor writes both stdout and stderr to *.log files that you can provide to Salesforce Customer Support or attach to a GitHub issue.

Plugin providers can also implement their own doctor diagnostic tests by listening to the "sf-doctor" event and running plugin specific tests that are then included in the doctor diagnostics log.

Examples for `doctor`

Run CLI doctor diagnostics:

```
sf doctor
```

Run CLI doctor diagnostics and the specified command, and write the debug output to a file:

```
sf doctor --command "force:org:list --all"
```

Run CLI doctor diagnostics for a specific plugin:

```
sf doctor --plugin @salesforce/plugin-source
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-c | --command COMMAND
Optional

Command to run in debug mode; results are written to a log file.

Type: option

-p | --plugin PLUGIN
Optional

Specific plugin on which to run diagnostics.

Type: option

-d | --output-dir OUTPUT-DIR
Optional

Directory to save all created files rather than the current working directory.

Type: option

-i | --create-issue
Optional

Create a new issue on our GitHub repo and attach all diagnostic results.

Type: boolean

`flow` Commands

Commands for testing flows

`flow get test`

Display test results for a specific asynchronous test run.

Description for `flow get test`

Provide a flow test run ID to display test results for an enqueued or completed asynchronous test run. The test run ID is displayed after running the "sf flow run test" command.

To see code coverage results, use the --code-coverage flag with --result-format. The output displays a high-level summary of the test run and the code coverage values for flow tests in your org. If you specify human-readable result format, use the --detailed-coverage flag to see detailed coverage results for each test method run.

Examples for `flow get test`

Display flow test results for your default org using a test run ID:

```
sf flow get test --test-run-id <test run id>
```

Similar to previous example, but output the result in JUnit format:

```
sf flow get test --test-run-id <test run id> --result-format junit
```

Also retrieve code coverage results and output in JSON format:

```
sf flow get test --test-run-id <test run id> --code-coverage --json
```

Specify a directory in which to save the test results from the org with the “me@my.org” username (rather than your default org):

```
sf flow get test --test-run-id <test run id> --code-coverage --output-dir <path to outputdir>
 --target-org me@my.org'
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-i | --test-run-id TEST-RUN-ID`
Required

ID of the test run.

Type: option

-c | --code-coverage
Optional

Retrieve code coverage results.

Type: boolean

--detailed-coverage
Optional

Not available for flow tests.

Type: boolean

-d | --output-dir OUTPUT-DIR
Optional

Directory in which to store test result files.

Type: option

-r | --result-format RESULT-FORMAT
Optional

Format of the test results.

Type: option

Permissible values are: human, tap, junit, json

Default value: human

--concise
Optional

Display only failed test results; works with human-readable output only.

Type: boolean

`flow run test`

Invoke flow tests in an org.

Description for `flow run test`

Specify which tests to run by using the --class-names flag followed by the names of the flows you want to test. For example, if you save a flow with the name Flow1, then use: --class-names Flow1.

To see code coverage results, use the --code-coverage flag with --result-format. The output displays a high-level summary of the test run and the code coverage values for classes in your org. If you specify human-readable result format, use the --detailed-coverage flag to see detailed coverage results for each test method run.

By default, "flow run test" runs asynchronously and immediately returns a test run ID. If you use the -–synchronous flag, you can use the --wait flag to specify the number of minutes to wait; if the tests finish in that timeframe, the command displays the results. If the tests haven't finished by the end of the wait time, the command displays a test run ID. Use the "flow get test --test-run-id" command to get the results.

You must have the "View All Data" org system permission to use this command. The permission is disabled by default and can be enabled only by a system administrator.

Examples for `flow run test`

Run all local tests in your default org:

```
sf flow run test --test-level RunLocalTests
```

Run all the Flow1 and Flow2 flow tests in the org with alias "scratchOrg":

```
sf flow run test --target-org scratchOrg --class-names Flow1 --class-names Flow2
```

Run specific Flow1 and Flow2 flow tests in your default org:

```
sf flow run test --tests Flow1.Test1 --tests Flow2.Test2 --test-level RunSpecifiedTests
```

Run all tests synchronously in your default org; the command waits to display the test results until all tests finish:

```
sf flow run test –synchronous
```

Run all local tests in the org with the username "me@my.org"; save the output to the specified directory:

```
sf flow run test --test-level RunLocalTests --output-dir /Users/susan/temp/cliOutput
--target-org me@my.org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-r | --result-format RESULT-FORMAT`
Optional

Format of the test results.

Type: option

Permissible values are: human, tap, junit, json

Default value: human

--concise
Optional

Display only failed test results; works with human-readable output only.

Type: boolean

-d | --output-dir OUTPUT-DIR
Optional

Directory in which to store test result files.

Type: option

-c | --code-coverage
Optional

Retrieve code coverage results.

Type: boolean

-y | --synchronous
Optional

Run flow tests for one flow synchronously; if not specified, tests are run asynchronously.

Type: boolean

-l | --test-level TEST-LEVEL
Optional

Level of tests to run; default is RunLocalTests.

Here's what the levels mean:

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages.

- RunAllTestsInOrg — All tests are run. The tests include all tests in your org, including tests of managed packages.

- RunSpecifiedTests - Only the tests that you specify with the --tests flag are run.

Type: option

Permissible values are: RunLocalTests, RunAllTestsInOrg, RunSpecifiedTests

-n | --class-names CLASS-NAMES
Optional

Flow names that contain flow tests to run.

Default is all flow tests. If you select --class-names, you can't specify --tests.

Type: option

-s | --suite-names SUITE-NAMES
Optional

Not available for flow tests.

Not available for flow tests.

Type: option

-t | --tests TESTS
Optional

Flow test names to run.

Default is all flow tests. If you specify --tests, you can't specify --class-names.

Type: option

`force` Commands

Legacy commands for backward compatibility.

`force data bulk delete`

Bulk delete records from an org using a CSV file. Uses Bulk API 1.0.

Description for `force data bulk delete`

The CSV file must have only one column ("Id") and then the list of record IDs you want to delete, one ID per line.

When you execute this command, it starts a job and one or more batches, displays their IDs, and then immediately returns control of the terminal to you by default. If you prefer to wait, set the --wait flag to the number of minutes; if it times out, the command outputs the IDs. Use the job and batch IDs to check the status of the job with the "sf force data bulk status" command. A single job can contain many batches, depending on the length of the CSV file.

Examples for `force data bulk delete`

Bulk delete Account records from your default org using the list of IDs in the "files/delete.csv" file:

```
sf force data bulk delete --sobject Account --file files/delete.csv
```

Bulk delete records from a custom object in an org with alias my-scratch and wait 5 minutes for the command to complete:

```
sf force data bulk delete --sobject MyObject__c --file files/delete.csv --wait 5 --target-org
 my-scratch
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-f | --file FILE
Required

CSV file that contains the IDs of the records to delete.

Type: option

-s | --sobject SOBJECT
Required

API name of the Salesforce object, either standard or custom, that you want to delete records from.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

force data bulk status

View the status of a bulk data load job or batch. Uses Bulk API 1.0.

Description for force data bulk status

Run this command using the job ID or batch ID returned from the "sf force data bulk delete" or "sf force data bulk upsert" commands.

Examples for force data bulk status

View the status of a bulk load job in your default org:

```
sf force data bulk status --job-id 750xx000000005sAAA
```

View the status of a bulk load job and a specific batches in an org with alias my-scratch:

```
sf force data bulk status --job-id 750xx000000005sAAA --batch-id 751xx000000005nAAA
--target-org my-scratch
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-b | --batch-id BATCH-ID
Optional

ID of the batch whose status you want to view; you must also specify the job ID.

Type: option

-i | --job-id JOB-ID
Required

ID of the job whose status you want to view.

Type: option

force data bulk upsert

Bulk upsert records to an org from a CSV file. Uses Bulk API 1.0.

Description for force data bulk upsert

An upsert refers to inserting a record into a Salesforce object if the record doesn't already exist, or updating it if it does exist.

When you execute this command, it starts a job and one or more batches, displays their IDs, and then immediately returns control of the terminal to you by default. If you prefer to wait, set the --wait flag to the number of minutes; if it times out, the command outputs the IDs. Use the job and batch IDs to check the status of the job with the "sf force data bulk status" command. A single job can contain many batches, depending on the length of the CSV file.

See "Prepare CSV Files" in the Bulk API Developer Guide for details on formatting your CSV file. (https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/datafiles_csv_preparing.htm)

By default, the job runs the batches in parallel, which we recommend. You can run jobs serially by specifying the --serial flag. But don't process data in serial mode unless you know this would otherwise result in lock timeouts and you can't reorganize your batches to avoid the locks.

Examples for `force data bulk upsert`

Bulk upsert records to the Contact object in your default org:

```
sf --sobject Contact --file files/contacts.csv --external-id Id
```

Bulk upsert records to a custom object in an org with alias my-scratch and wait 5 minutes for the command to complete:

```
sf force data bulk upsert --sobject MyObject__c --file files/file.csv --external-id
MyField__c --wait 5 --target-org my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-i | --external-id EXTERNAL-ID`
Required

Name of the external ID field, or the Id field.

Type: option

`-f | --file FILE`
Required

CSV file that contains the records to upsert.

Type: option

`-s | --sobject SOBJECT`
Required

API name of the Salesforce object, either standard or custom, that you want to upsert records to.

Type: option

`-w | --wait WAIT`
Optional

Number of minutes to wait for the command to complete before displaying the results.

Type: option

-r | --serial

Optional

Run batches in serial mode.

Type: boolean

force lightning lwc test create

Description for force lightning lwc test create

creates a Lightning web component test file with boilerplate code inside a __tests__ directory.

Examples for force lightning lwc test create

```
$ sfdx force:lightning:lwc:test:create -f force-app/main/default/lwc/myButton/myButton.js
```

Flags

--json

Optional

format output as json

Type: boolean

--loglevel LOGLEVEL

Optional

logging level for this command invocation

Type: enum

Permissible values are: trace, debug, info, warn, error, fatal, TRACE, DEBUG, INFO, WARN, ERROR, FATAL

Default value: warn

-f | --filepath FILEPATH

Required

path to Lightning web component .js file to create a test for

Type: string

force lightning lwc test run

Description for force lightning lwc test run

invokes Lightning Web Components Jest unit tests.

Examples for `force lightning lwc test run`

```
$ sfdx force:lightning:lwc:test:run
```

```
$ sfdx force:lightning:lwc:test:run -w
```

Flags

`--json`
Optional

format output as json

Type: boolean

`--loglevel LOGLEVEL`
Optional

logging level for this command invocation

Type: enum

Permissible values are: trace, debug, info, warn, error, fatal, TRACE, DEBUG, INFO, WARN, ERROR, FATAL

Default value: warn

`-d | --debug`
Optional

run tests in debug mode

Type: boolean

`--watch`
Optional

run tests in watch mode

Type: boolean

`force lightning lwc test setup`

Description for `force lightning lwc test setup`

install Jest unit testing tools for Lightning Web Components.

Examples for `force lightning lwc test setup`

```
$ sfdx force:lightning:lwc:test:setup
```

Flags

`--json`
Optional

format output as json

Type: boolean

--loglevel LOGLEVEL

Optional

logging level for this command invocation

Type: enum

Permissible values are: trace, debug, info, warn, error, fatal, TRACE, DEBUG, INFO, WARN, ERROR, FATAL

Default value: warn

info Commands

Access Salesforce CLI information from the command line.

info releasenotes display

Display Salesforce CLI release notes on the command line.

Description for info releasenotes display

By default, this command displays release notes for the currently installed CLI version on your computer. Use the --version flag to view release notes for a different release.

Examples for info releasenotes display

Display release notes for the currently installed CLI version:

```
sf info releasenotes display
```

Display release notes for CLI version 7.120.0:

```
sf info releasenotes display --version 7.120.0
```

Display release notes for the CLI version that corresponds to a tag (stable, stable-rc, latest, latest-rc, rc):

```
sf info releasenotes display --version latest
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-v | --version VERSION
Optional

CLI version or tag for which to display release notes.

Type: option

Aliases for `info releasenotes display`

```
whatsnew
```

`lightning` Commands

Work with Lightning Web and Aura components.

lightning generate app
Generate a Lightning App.

lightning generate component
Generate a bundle for an Aura component or a Lightning web component.

lightning generate event
Generate a Lightning Event.

lightning generate interface
Generate a Lightning Interface.

lightning generate test
Generate a Lightning test.

`lightning generate app`

Generate a Lightning App.

Description for `lightning generate app`

Generates a Lightning App bundle in the specified directory or the current working directory. The bundle consists of multiple files in a folder with the designated name.

Examples for `lightning generate app`

Generate the metadata files for a Lightning app bundle called "myapp" in the current directory:

```
sf lightning generate app --name myapp
```

Similar to the previous example, but generate the files in the "force-app/main/default/aura" directory:

```
sf lightning generate app --name myapp --output-dir force-app/main/default/aura
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Name of the generated Lightning App.

The name can be up to 40 characters and must start with a letter.

Type: option

-t | --template TEMPLATE
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: DefaultLightningApp

Default value: DefaultLightningApp

-d | --output-dir OUTPUT-DIR
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `lightning generate app`

```
force:lightning:app:create
```

`lightning generate component`

Generate a bundle for an Aura component or a Lightning web component.

Description for `lightning generate component`

Generates the bundle in the specified directory or the current working directory. The bundle consists of multiple files in a directory with the designated name. Lightning web components are contained in the directory with name "lwc", Aura components in "aura".

To generate a Lightning web component, pass "--type lwc" to the command. If you don't specify --type, Salesforce CLI generates an Aura component by default.

Examples for `lightning generate component`

Generate the metadata files for an Aura component bundle in the current directory:

```
sf lightning generate component --name mycomponent
```

Generate a Lightning web component bundle in the current directory:

```
sf lightning generate component --name mycomponent --type lwc
```

Generate an Aura component bundle in the "force-app/main/default/aura" directory:

```
sf lightning generate component --name mycomponent --output-dir force-app/main/default/aura
```

Generate a Lightning web component bundle in the "force-app/main/default/lwc" directory:

```
sf lightning generate component --name mycomponent --type lwc --output-dir
force-app/main/default/lwc
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated Lightning Component.

The name can be up to 40 characters and must start with a letter.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: default, analyticsDashboard, analyticsDashboardWithStep

Default value: default

-d | --output-dir OUTPUT-DIR
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

--type TYPE
Optional

Type of the component bundle.

Type: option

Permissible values are: aura, lwc

Default value: aura

Aliases for `lightning generate component`

```
force:lightning:component:create
```

`lightning generate event`

Generate a Lightning Event.

Description for `lightning generate event`

Generates a Lightning Event bundle in the specified directory or the current working directory. The bundle consists of multiple files in a folder with the designated name.

Examples for `lightning generate event`

Generate the metadata files for a Lightning event bundle called "myevent" in the current directory:

```
sf lightning generate event --name myevent
```

Similar to previous example, but generate the files in the "force-app/main/default/aura" directory:

```
sf lightning generate event --name myevent --output-dir force-app/main/default/aura
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-n | --name NAME

Required

Name of the generated Lightning Event.

The name can be up to 40 characters and must start with a letter.

Type: option

-t | --template TEMPLATE

Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: DefaultLightningEvt

Default value: DefaultLightningEvt

-d | --output-dir OUTPUT-DIR

Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `lightning generate event`

```
force:lightning:event:create
```

`lightning generate interface`

Generate a Lightning Interface.

Description for `lightning generate interface`

Generates a Lightning Interface bundle in the specified directory or the current working directory. The bundle consists of multiple files in a folder with the designated name.

Examples for `lightning generate interface`

Generate the metadata files for a Lightning interface bundle called "myinterface" in the current directory:

```
sf lightning generate interface --name myinterface
```

Similar to the previous example but generate the files in the "force-app/main/default/aura" directory:

```
sf lightning generate interface --name myinterface --output-dir force-app/main/default/aura
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated Lightning Interface.

The name can be up to 40 characters and must start with a letter.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: DefaultLightningIntf

Default value: DefaultLightningIntf

`-d | --output-dir OUTPUT-DIR`
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `lightning generate interface`

```
force:lightning:interface:create
```

`lightning generate test`

Generate a Lightning test.

Description for `lightning generate test`

Generates the test in the specified directory or the current working directory. The .resource file and associated metadata file are generated.

Examples for `lightning generate test`

Generate the metadata files for the Lightning test called MyLightningTest in the current directory:

```
sf lightning generate test --name MyLightningTest
```

Similar to the previous example but generate the files in the "force-app/main/default/lightningTests" directory:

```
sf lightning generate test --name MyLightningTest --output-dir
force-app/main/default/lightningTests
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated Lightning Test.

Name of the new Lightning test; can be up to 40 characters and must start with a letter.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: DefaultLightningTest

Default value: DefaultLightningTest

-d | --output-dir OUTPUT-DIR

Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `lightning generate test`

```
force:lightning:test:create
```

`org` Commands

Commands to create and manage orgs and scratch org users.

org assign permset
Assign a permission set to one or more org users.

org assign permsetlicense
Assign a permission set license to one or more org users.

org create sandbox
Create a sandbox org.

org create scratch
Create a scratch org.

org create shape
Create a scratch org configuration (shape) based on the specified source org.

org create snapshot
Create a snapshot of a scratch org.

org create user
Create a user for a scratch org.

org delete sandbox
Delete a sandbox.

org delete scratch
Delete a scratch org.

org delete shape
Delete all org shapes for a target org.

org delete snapshot
Delete a scratch org snapshot.

org disable tracking
Prevent Salesforce CLI from tracking changes in your source files between your project and an org.

org display
Display information about an org.

org display user
Display information about a Salesforce user.

org enable tracking
Allow Salesforce CLI to track changes in your source files between your project and an org.

org generate password
Generate a random password for scratch org users.

org get snapshot
Get details about a scratch org snapshot.

org list
List all orgs you've created or authenticated to.

org list auth
List authorization information about the orgs you created or logged into.

org list limits
Display information about limits in your org.

org list metadata
List the metadata components and properties of a specified type.

org list metadata-types
Display details about the metadata types that are enabled for your org.

org list shape
List all org shapes you've created.

org list snapshot
List scratch org snapshots.

org list sobject record-counts
Display record counts for the specified standard or custom objects.

org list users
List all locally-authenticated users of an org.

org login access-token
Authorize an org using an existing Salesforce access token.

org login device
Authorize an org using a device code.

org login jwt
Log in to a Salesforce org using a JSON web token (JWT).

org login sfdx-url
Authorize an org using a Salesforce DX authorization URL stored in a file or through standard input (stdin).

org login web
Log in to a Salesforce org using the web server flow.

org logout
Log out of a Salesforce org.

org open
Open your default scratch org, or another specified org, in a browser.

org open agent (Beta)
Open an agent in your org's Agent Builder UI in a browser.

org refresh sandbox
Refresh a sandbox org using the sandbox name.

org resume sandbox
Check the status of a sandbox creation, and log in to it if it's ready.

org resume scratch
Resume the creation of an incomplete scratch org.

`org assign permset`

Assign a permission set to one or more org users.

Description for `org assign permset`

To specify an alias for the --target-org or --on-behalf-of flags, use the CLI username alias, such as the one you set with the "alias set" command. Don't use the value of the Alias field of the User Salesforce object for the org user.

To assign multiple permission sets, either set multiple --name flags or a single --name flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --on-behalf-of.

Examples for `org assign permset`

Assign two permission sets called DreamHouse and CloudHouse to original admin user of your default org:

```
sf org assign permset --name DreamHouse --name CloudHouse
```

Assign the Dreamhouse permission set to the original admin user of the org with alias "my-scratch":

```
sf org assign permset --name DreamHouse --target-org my-scratch
```

Assign the Dreamhouse permission set to the specified list of users of your default org:

```
sf org assign permset --name DreamHouse --on-behalf-of user1@my.org --on-behalf-of user2
--on-behalf-of user
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Permission set to assign.

Type: option

-b | --on-behalf-of ON-BEHALF-OF
Optional

Username or alias to assign the permission set to.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

`org assign permsetlicense`

Assign a permission set license to one or more org users.

Description for `org assign permsetlicense`

To specify an alias for the --target-org or --on-behalf-of flags, use the CLI username alias, such as the one you set with the "alias set" command. Don't use the value of the Alias field of the User Salesforce object for the org user.

To assign multiple permission sets, either set multiple --name flags or a single --name flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --on-behalf-of.

Examples for `org assign permsetlicense`

Assign the DreamHouse permission set license to original admin user of your default org:

```
sf org assign permsetlicense --name DreamHouse
```

Assign two permission set licenses to the original admin user of the org with alias "my-scratch":

```
sf org assign permsetlicense --name DreamHouse --name CloudHouse --target-org my-scratch
```

Assign the Dreamhouse permission set license to the specified list of users of your default org:

```
sf org assign permsetlicense --name DreamHouse --on-behalf-of user1@my.org --on-behalf-of
 user2 --on-behalf-of user3
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Name of the permission set license to assign.

Type: option

-b | --on-behalf-of ON-BEHALF-OF
Optional

Usernames or alias to assign the permission set license to.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

org create sandbox

Create a sandbox org.

Description for org create sandbox

There are two ways to create a sandbox org: specify a definition file that contains the sandbox options or use the --name and --license-type flags to specify the two required options. If you want to set an option other than name or license type, such as apexClassId, you must use a definition file.

You can also use this command to clone an existing sandbox. Use the --source-sandbox-name flag to specify the existing sandbox name and the --name flag to the name of the new sandbox.

Examples for `org create sandbox`

Create a sandbox org using a definition file and give it the alias "MyDevSandbox". The production org that contains the sandbox license has the alias "prodOrg".

```
sf org create sandbox --definition-file config/dev-sandbox-def.json --alias MyDevSandbox
--target-org prodOrg
```

Create a sandbox org by directly specifying its name and type of license (Developer) instead of using a definition file. Set the sandbox org as your default.

```
sf org create sandbox --name mysandbox --license-type Developer --alias MyDevSandbox
--target-org prodOrg --set-default
```

Clone the existing sandbox with name "ExistingSandbox" and name the new sandbox "NewClonedSandbox". Set the new sandbox as your default org. Wait for 30 minutes for the sandbox creation to complete.

```
sf org create sandbox --source-sandbox-name ExistingSandbox --name NewClonedSandbox
--target-org prodOrg --alias MyDevSandbox --set-default --wait 30
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-f | --definition-file DEFINITION-FILE`
Optional

Path to a sandbox definition file.

The sandbox definition file is a blueprint for the sandbox. You can create different definition files for each sandbox type that you use in the development process. See <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_sandbox_definition.htm> for all the options you can specify in the definition file.

Type: option

`-s | --set-default`
Optional

Set the sandbox org as your default org.

Type: boolean

`-a | --alias ALIAS`
Optional

Alias for the sandbox org.

When you create a sandbox, the generated usernames are based on the usernames present in the production org. To ensure uniqueness, the new usernames are appended with the name of the sandbox. For example, the username "user@example.com" in

the production org results in the username "user@example.com.mysandbox" in a sandbox named "mysandbox". When you set an alias for a sandbox org, it's assigned to the resulting username of the user running this command.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the sandbox org to be ready.

If the command continues to run after the wait period, the CLI returns control of the terminal to you and displays the "sf org resume sandbox" command you run to check the status of the create. The displayed command includes the job ID for the running sandbox creation.

Type: option

Default value: 30 minutes

-i | --poll-interval POLL-INTERVAL
Optional

Number of seconds to wait between retries.

Type: option

Default value: 30 seconds

--async
Optional

Request the sandbox creation, but don't wait for it to complete.

The command immediately displays the job ID and returns control of the terminal to you. This way, you can continue to use the CLI. To check the status of the sandbox creation, run "sf org resume sandbox".

Type: boolean

-n | --name NAME
Optional

Name of the sandbox org.

The name must be a unique alphanumeric string (10 or fewer characters) to identify the sandbox. You can’t reuse a name while a sandbox is in the process of being deleted.

Type: option

--source-sandbox-name SOURCE-SANDBOX-NAME
Optional

Name of the sandbox org to clone.

The value of --source-sandbox-name must be an existing sandbox. The existing sandbox, and the new sandbox specified with the --name flag, must both be associated with the production org (--target-org) that contains the sandbox licenses.

You can specify either --source-sandbox-name or --source-id when cloning an existing sandbox, but not both.

Type: option

--source-id SOURCE-ID
Optional

ID of the sandbox org to clone.

The value of --source-id must be an existing sandbox. The existing sandbox, and the new sandbox specified with the --name flag, must both be associated with the production org (--target-org) that contains the sandbox licenses.

You can specify either --source-sandbox-name or --source-id when cloning an existing sandbox, but not both.

Type: option

-l | --license-type LICENSE-TYPE
Optional

Type of sandbox license.

Type: option

Permissible values are: Developer, Developer_Pro, Partial, Full

-o | --target-org TARGET-ORG
Required

Username or alias of the production org that contains the sandbox license.

When it creates the sandbox org, Salesforce copies the metadata, and optionally data, from your production org to the new sandbox org.

Type: option

--no-prompt
Optional

Don't prompt for confirmation about the sandbox configuration.

Type: boolean

--no-track-source
Optional

Do not use source tracking for this sandbox.

We recommend you enable source tracking in Developer and Developer Pro sandbox, which is why it's the default behavior. Source tracking allows you to track the changes you make to your metadata, both in your local project and in the sandbox, and to detect any conflicts between the two.

To disable source tracking in the new sandbox, specify the --no-track-source flag. The main reason to disable source tracking is for performance. For example, while you probably want to deploy metadata and run Apex tests in your CI/CD jobs, you probably don't want to incur the costs of source tracking (checking for conflicts, polling the SourceMember object, various file system operations.) This is a good use case for disabling source tracking in the sandbox.

Type: boolean

Aliases for `org create sandbox`

```
env:create:sandbox
```

`org create scratch`

Create a scratch org.

Description for `org create scratch`

There are four ways to create a scratch org:

* Specify a definition file that contains the scratch org options.

* Use the --edition flag to specify the one required option; this method doesn't require a defintion file.

* Use the --snapshot flag to create a scratch org from a snapshot. Snapshots are a point-in-time copy of a scratch org; you create a snapshot with the "sf org create snapshot" command.

* Use the --source-org flag to create a scratch org from an org shape. Org shapes mimic the baseline setup of a source org without the extraneous data and metadata; you create an org shape with the "sf org create shape" command.

The --edition, --snapshot, and --source-org flags are mutually exclusive, which means if you specify one, you can't also specify the others.

For any of the methods, you can also use these flags; if you use them with --definition-file, they override their equivalent option in the scratch org definition file:

* --description

* --name (equivalent to the "orgName" option)

* --username

* --release

* --admin-email (equivalent to the "adminEmail" option)

If you want to set options such as org features or settings, you must use a definition file.

You must specify a Dev Hub to create a scratch org, either with the --target-dev-hub flag or by setting your default Dev Hub with the target-dev-hub configuration variable.

Examples for `org create scratch`

Create a Developer edition scratch org using your default Dev Hub and give the scratch org an alias:

```
sf org create scratch --edition developer --alias my-scratch-org
```

Create a scratch org with a definition file. Specify the Dev Hub using its alias, set the scratch org as your default, and specify that it expires in 3 days:

```
sf org create scratch --target-dev-hub MyHub --definition-file
config/project-scratch-def.json --set-default --duration-days 3
```

Create a preview Enterprise edition scratch org; for use only during Salesforce release transition periods:

```
sf org create scratch --edition enterprise --alias my-scratch-org --target-dev-hub MyHub
--release preview
```

Create a scratch org from a snapshot called "NightlyBranch"; be sure you specify the same Dev Hub org associated with the snapshot. We recommend you increase the --wait time because creating a scratch org from a snapshot can take a while:

```
sf org create scratch --alias my-scratch-org --target-dev-hub MyHub --snapshot NightlyBranch
 --wait 10
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

-a | --alias ALIAS
Optional

Alias for the scratch org.

New scratch orgs include one administrator by default. The admin user's username is auto-generated and looks something like test-wvkpnfm5z113@example.com. When you set an alias for a new scratch org, it's assigned this username.

Type: option

--async
Optional

Request the org, but don't wait for it to complete.

The command immediately displays the job ID and returns control of the terminal to you. This way, you can continue to use the CLI. To resume the scratch org creation, run "sf org resume scratch".

Type: boolean

-d | --set-default
Optional

Set the scratch org as your default org

Type: boolean

-f | --definition-file DEFINITION-FILE
Optional

Path to a scratch org definition file.

The scratch org definition file is a blueprint for the scratch org. It mimics the shape of an org that you use in the development life cycle, such as acceptance testing, packaging, or production. See <https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_def_file.htm> for all the option you can specify in the definition file.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org.

Overrides the value of the target-dev-hub configuration variable, if set.

Type: option

-c | --no-ancestors
Optional

Don't include second-generation managed package (2GP) ancestors in the scratch org.

Type: boolean

-e | --edition EDITION
Optional

Salesforce edition of the scratch org. Overrides the value of the "edition" option in the definition file, if set.

The editions that begin with "partner-" are available only if the Dev Hub org is a Partner Business Org.

Type: option

Permissible values are: developer, enterprise, group, professional, partner-developer, partner-enterprise, partner-group, partner-professional

-s | --snapshot SNAPSHOT
Optional

Name of the snapshot to use when creating this scratch org. Overrides the value of the "snapshot" option in the defintion file, if set.

To view the names of the available snapshots for a given Dev Hub org, run the "sf org list snapshot" command.

Type: option

-m | --no-namespace
Optional

Create the scratch org with no namespace, even if the Dev Hub has a namespace.

Type: boolean

-y | --duration-days DURATION-DAYS
Optional

Number of days before the org expires.

Type: option

Default value: 7 days

-w | --wait WAIT
Optional

Number of minutes to wait for the scratch org to be ready.

If the command continues to run after the wait period, the CLI returns control of the terminal to you and displays the job ID. To resume the scratch org creation, run the org resume scratch command and pass it the job ID.

Type: option

Default value: 5 minutes

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --client-id CLIENT-ID
Optional

Consumer key of the Dev Hub connected app.

Type: option

-t | --track-source
Optional

Use source tracking for this scratch org. Set --no-track-source to disable source tracking.

We recommend you enable source tracking in scratch orgs, which is why it's the default behavior. Source tracking allows you to track the changes you make to your metadata, both in your local project and in the scratch org, and to detect any conflicts between the two.

To disable source tracking in the new scratch org, specify the --no-track-source flag. The main reason to disable source tracking is for performance. For example, while you probably want to deploy metadata and run Apex tests in your CI/CD jobs, you probably

don't want to incur the costs of source tracking (checking for conflicts, polling the SourceMember object, various file system operations.) This is a good use case for disabling source tracking in the scratch org.

Type: boolean

Default value: true

--username USERNAME

Optional

Username of the scratch org admin user. Overrides the value of the "username" option in the definition file, if set.

The username must be unique within the entire scratch org and sandbox universe. You must add your own logic to ensure uniqueness.

Omit this flag to have Salesforce generate a unique username for your org.

Type: option

--description DESCRIPTION

Optional

Description of the scratch org in the Dev Hub. Overrides the value of the "description" option in the definition file, if set.

Type: option

--name NAME

Optional

Name of the org, such as "Acme Company". Overrides the value of the "orgName" option in the definition file, if set.

Type: option

--release RELEASE

Optional

Release of the scratch org as compared to the Dev Hub release.

By default, scratch orgs are on the same release as the Dev Hub. During Salesforce release transition periods, you can override this default behavior and opt in or out of the new release.

Type: option

Permissible values are: preview, previous

--admin-email ADMIN-EMAIL

Optional

Email address that will be applied to the org's admin user. Overrides the value of the "adminEmail" option in the definition file, if set.

Type: option

--source-org SOURCE-ORG

Optional

15-character ID of the org shape that the new scratch org is based on. Overrides the value of the "sourceOrg" option in the definition file, if set.

To view the names of the available org shapes for a given Dev Hub org, run the "sf org list shape" command.

Type: option

Aliases for `org create scratch`

```
env:create:scratch
```

`org create shape`

Create a scratch org configuration (shape) based on the specified source org.

Description for `org create shape`

Scratch org shapes mimic the baseline setup (features, limits, edition, and Metadata API settings) of a source org without the extraneous data and metadata.

Run "sf org list shape" to view the available org shapes and their IDs.

To create a scratch org from an org shape, include the "sourceOrg" property in the scratch org definition file and set it to the org ID of the source org. Then create a scratch org with the "sf org create scratch" command.

Examples for `org create shape`

Create an org shape for the source org with alias SourceOrg:

```
sf org create shape --target-org SourceOrg
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org create shape`

```
force:org:shape:create
```

`org create snapshot`

Create a snapshot of a scratch org.

Description for `org create snapshot`

A snapshot is a point-in-time copy of a scratch org. The copy is referenced by its unique name in a scratch org definition file.

Use "sf org get snapshot" to get details, including status, about a snapshot creation request.

To create a scratch org from a snapshot, include the "snapshot" option (instead of "edition") in the scratch org definition file and set it to the name of the snapshot. Then use "sf org create scratch" to create the scratch org.

Examples for `org create snapshot`

Create a snapshot called "Dependencies" using the source scratch org ID and your default Dev Hub org:

```
sf org create snapshot --source-org 00Dxx0000000000 --name Dependencies --description
'Contains PackageA v1.1.0'
```

Create a snapshot called "NightlyBranch" using the source scratch org username and a Dev Hub org with alias NightlyDevHub:

```
sf org create snapshot --source-org myuser@myorg --name NightlyBranch --description 'Contains
 PkgA v2.1.0 and PkgB 3.3.0' --target-dev-hub NightlyDevHub
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-o | --source-org SOURCE-ORG`
Required

ID or locally authenticated username or alias of scratch org to snapshot.

Type: option

`-n | --name NAME`
Required

Unique name of snapshot.

Type: option

-d | --description DESCRIPTION

Optional

Description of snapshot.

Use this description to document the contents of the snapshot. We suggest that you include a reference point, such as a version control system tag or commit ID.

Type: option

Aliases for `org create snapshot`

```
force:org:snapshot:create
```

`org create user`

Create a user for a scratch org.

Description for `org create user`

A scratch org includes one administrator user by default. For testing purposes, however, you sometimes need to create additional users.

The easiest way to create a user is to let this command assign default or generated characteristics to the new user. If you want to customize your new user, create a definition file and specify it with the --definition-file flag. In the file, you can include all the User sObject (SSalesforce object) fields and Salesforce DX-specific options, as described in "User Definition File for Customizing a Scratch Org User" (https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_users_def_file.htm). You can also specify these options on the command line.

If you don't customize your new user, this command creates a user with the following default characteristics:

* The username is the existing administrator's username prepended with a timestamp, such as 1505759162830_test-wvkpnfm5z113@example.com.

* The user's profile is Standard User.

* The values of the required fields of the User sObject are the corresponding values of the administrator user.

* The user has no password.

Use the --set-alias flag to assign a simple name to the user that you can reference in later CLI commands. This alias is local and different from the Alias field of the User sObject record of the new user, which you set in the Setup UI.

When this command completes, it displays the new username and user ID. Run the "org display user" command to get more information about the new user.

After the new user has been created, Salesforce CLI automatically authenticates it to the scratch org so the new user can immediately start using the scratch org. The CLI uses the same authentication method that was used on the associated Dev Hub org. Due to Hyperforce limitations, the scratch org user creation fails if the Dev Hub authentication used the JWT flow and the scratch org is on Hyperforce. For this reason, if you plan to create scratch org users, authenticate to the Dev Hub org with either the "org login web" or "org login sfdx-url" command, and not "org login jwt".

For more information about user limits, defaults, and other considerations when creating a new scratch org user, see https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_users.htm.

Examples for `org create user`

Create a user for your default scratch org and let this command generate a username, user ID, and other characteristics:

```
sf org create user
```

Create a user with alias "testuser1" using a user definition file. Set the "profileName" option to "Chatter Free User", which overrides the value in the defintion file if it also exists there. Create the user for the scratch org with alias "my-scratch":

```
sf org create user --set-alias testuser1 --definition-file config/project-user-def.json
profileName='Chatter Free User' --target-org my-scratch
```

Create a user by specifying the username, email, and perm set assignment at the command line; command fails if the username already exists in Salesforce:

```
sf org create user username=testuser1@my.org email=me@my.org permsets=DreamHouse
```

Create a user with a definition file, set the email value as specified (overriding any value in the definition file), and generate a password for the user. If the username in the definition file isn't unique, the command appends the org ID to make it unique:

```
sf org create user --definition-file config/project-user-def.json email=me@my.org
generatepassword=true --set-unique-username
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-a | --set-alias SET-ALIAS`
Optional

Set an alias for the created username to reference in other CLI commands.

Type: option

`-f | --definition-file DEFINITION-FILE`
Optional

File path to a user definition file for customizing the new user.

The user definition file uses JSON format and can include any Salesforce User sObject field and Salesforce DX-specific options. See https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_users_def_file.htm for more information.

Type: option

`-s | --set-unique-username`
Optional

Force the username, if specified in the definition file or at the command line, to be unique by appending the org ID.

The new user's username must be unique across all Salesforce orgs and in the form of an email address. If you let this command generate a username for you, it's guaranteed to be unique. If you specify an existing username in a definition file, the command fails. Set this flag to force the username to be unique; as a result, the username might be different than what you specify in the definition file.

Type: boolean

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org create user`

```
force:user:create
```

`org delete sandbox`

Delete a sandbox.

Description for `org delete sandbox`

Salesforce CLI marks the org for deletion in the production org that contains the sandbox licenses and then deletes all local references to the org from your computer.

Specify a sandbox with either the username you used when you logged into it, or the alias you gave the sandbox when you created it. Run "sf org list" to view all your orgs, including sandboxes, and their aliases.

Both the sandbox and the associated production org must already be authenticated with the CLI to successfully delete the sandbox.

Examples for `org delete sandbox`

Delete a sandbox with alias my-sandbox:

```
sf org delete sandbox --target-org my-sandbox
```

Specify a username instead of an alias:

```
sf org delete sandbox --target-org myusername@example.com.qa
```

Delete the sandbox without prompting to confirm:

```
sf org delete sandbox --target-org my-sandbox --no-prompt
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-p | --no-prompt
Optional

Don't prompt the user to confirm the deletion.

Type: boolean

Aliases for `org delete sandbox`

```
env:delete:sandbox
```

`org delete scratch`

Delete a scratch org.

Description for `org delete scratch`

Salesforce CLI marks the org for deletion in the Dev Hub org and then deletes all local references to the org from your computer.

Specify a scratch org with either the username or the alias you gave the scratch org when you created it. Run "sf org list" to view all your orgs, including scratch orgs, and their aliases.

Examples for `org delete scratch`

Delete a scratch org with alias my-scratch-org:

```
sf org delete scratch --target-org my-scratch-org
```

Specify a username instead of an alias:

```
sf org delete scratch --target-org test-123456-abcdefg@example.com
```

Delete the scratch org without prompting to confirm :

```
sf org delete scratch --target-org my-scratch-org --no-prompt
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-p | --no-prompt
Optional

Don't prompt the user to confirm the deletion.

Type: boolean

Aliases for `org delete scratch`

```
env:delete:scratch
```

org delete shape

Delete all org shapes for a target org.

Description for `org delete shape`

A source org can have only one active org shape. If you try to create an org shape for a source org that already has one, the previous shape is marked inactive and replaced by a new active shape. If you don't want to create scratch orgs based on this shape, you can delete the org shape.

Examples for `org delete shape`

Delete all org shapes for the source org with alias SourceOrg:

```
sf org delete shape --target-org SourceOrg
```

Delete all org shapes without prompting:

```
sf org delete shape --target-org SourceOrg --no-prompt
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-p | --no-prompt
Optional

Don't prompt for confirmation.

Type: boolean

Aliases for `org delete shape`

```
force:org:shape:delete
```

`org delete snapshot`

Delete a scratch org snapshot.

Description for `org delete snapshot`

Dev Hub admins can delete any snapshot. Users can delete only their own snapshots, unless a Dev Hub admin gives the user Modify All permission, which works only with the Salesforce license.

Examples for `org delete snapshot`

Delete a snapshot from the default Dev Hub using the snapshot ID:

```
sf org delete snapshot --snapshot 0Oo...
```

Delete a snapshot from the specified Dev Hub using the snapshot name:

```
sf org delete snapshot --snapshot BaseSnapshot --target-dev-hub SnapshotDevHub
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-s | --snapshot SNAPSHOT
Required

Name or ID of snapshot to delete.

The IDs of scratch org snapshots start with 0Oo.

Type: option

-p | --no-prompt
Optional

Don't prompt the user to confirm the deletion.

Type: boolean

Aliases for `org delete snapshot`

```
force:org:snapshot:delete
```

`org disable tracking`

Prevent Salesforce CLI from tracking changes in your source files between your project and an org.

Description for `org disable tracking`

Disabling source tracking has no direct effect on the org, it affects only your local environment. Specifically, Salesforce CLI stores the setting in the org's local configuration file so that no source tracking operations are executed when working with the org.

Examples for `org disable tracking`

Disable source tracking for an org with alias "myscratch":

```
sf org disable tracking --target-org myscratch
```

Disable source tracking for an org using a username:

```
sf org disable tracking --target-org you@example.com
```

Disable source tracking for your default org:

```
sf org disable tracking
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

org display

Display information about an org.

Description for org display

Output includes your access token, client Id, connected status, org ID, instance URL, username, and alias, if applicable.

Use --verbose to include the SFDX auth URL. WARNING: The SFDX auth URL contains sensitive information, such as a refresh token that can be used to access an org. Don't share or distribute this URL or token.

Including --verbose displays the sfdxAuthUrl property only if you authenticated to the org using "org login web" (not "org login jwt").

Examples for org display

Display information about your default org:

```
$ sf org display
```

Display information, including the sfdxAuthUrl property, about the org with alias TestOrg1:

```
$ sf org display --target-org TestOrg1 --verbose
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

--verbose
Optional

Display the sfdxAuthUrl property.

Type: boolean

Aliases for `org display`

```
force:org:display
```

`org display user`

Display information about a Salesforce user.

Description for `org display user`

Output includes the profile name, org ID, access token, instance URL, login URL, and alias if applicable. The displayed alias is local and different from the Alias field of the User sObject record of the new user, which you set in the Setup UI.

Examples for `org display user`

Display information about the admin user of your default scratch org:

```
sf org display user
```

Display information about the specified user and output in JSON format:

```
sf org display user --target-org me@my.org --json
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org display user`

```
force:user:display
```

`org enable tracking`

Allow Salesforce CLI to track changes in your source files between your project and an org.

Description for `org enable tracking`

Enabling source tracking has no direct effect on the org, it affects only your local environment. Specifically, Salesforce CLI stores the setting in the org's local configuration file so that source tracking operations are executed when working with the org.

This command throws an error if the org doesn't support tracking. Examples of orgs that don't support source tracking include Developer Edition orgs, production orgs, Partial Copy sandboxes, and Full sandboxes.

Examples for `org enable tracking`

Enable source tracking for an org with alias "myscratch":

```
sf org enable tracking --target-org myscratch
```

Enable source tracking for an org using a username:

```
sf org enable tracking --target-org you@example.com
```

Enable source tracking for your default org:

```
sf org enable tracking
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`org generate password`

Generate a random password for scratch org users.

Description for `org generate password`

By default, new scratch orgs contain one admin user with no password. Use this command to generate or change a password for this admin user. After it's set, you can't unset a password, you can only change it.

You can also use the --on-behalf-of flag to generate a password for a scratch org user that you've created locally with the "org create user" command. This command doesn't work for users you created in the scratch org using Setup.

To change the password strength, set the --complexity flag to a value between 0 and 5. Each value specifies the types of characters used in the generated password:

0 - lower case letters only

1 - lower case letters and numbers only

2 - lower case letters and symbols only

3 - lower and upper case letters and numbers only

4 - lower and upper case letters and symbols only

5 - lower and upper case letters and numbers and symbols only

To see a password that was previously generated, run "org display user".

Examples for `org generate password`

Generate a password for the original admin user of your default scratch org:

```
sf org generate password
```

Generate a password that contains 12 characters for the original admin user of the scratch org with alias "my-scratch":

```
sf org generate password --length 12 --target-org my-scratch
```

Generate a password for your default scratch org admin user that uses lower and upper case letters and numbers only:

```
sf org generate password --complexity 3
```

Generate a password for the specified users in the default scratch org; these users must have been created locally with the "org create user" command:

```
sf org generate password --on-behalf-of user1@my.org --on-behalf-of user2@my.org
--on-behalf-of user3@my.org
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-b | --on-behalf-of ON-BEHALF-OF
Optional

Comma-separated list of usernames or aliases to assign the password to; must have been created locally with the "org create user" command.

Type: option

-l | --length LENGTH
Optional

Number of characters in the generated password; valid values are between 8 and 100.

Type: option

Default value: 13

-c | --complexity COMPLEXITY
Optional

Level of password complexity or strength; the higher the value, the stronger the password.

Type: option

Default value: 5

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

`org get snapshot`

Get details about a scratch org snapshot.

Description for `org get snapshot`

Snapshot creation can take a while. Use this command with the snapshot name or ID to check its creation status. After the status changes to Active, you can use the snapshot to create scratch orgs.

To create a snapshot, use the "sf org create snapshot" command. To retrieve a list of all snapshots, use "sf org list snapshot".

Examples for `org get snapshot`

Get snapshot details using its ID and the default Dev Hub org:

```
sf org get snapshot --snapshot 0Oo...
```

Get snapshot details using its name from a Dev Hub org with alias SnapshotDevHub:

```
sf org get snapshot --snapshot Dependencies --target-dev-hub SnapshotDevHub
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-s | --snapshot SNAPSHOT`
Required

Name or ID of snapshot to retrieve.

The IDs of scratch org snapshots start with 0Oo.

Type: option

Aliases for `org get snapshot`

```
force:org:snapshot:get
```

`org list`

List all orgs you've created or authenticated to.

Examples for `org list`

List all orgs you've created or authenticated to:

```
$ sf org list
```

List all orgs, including expired, deleted, and unknown-status orgs; don't include the connection status:

```
$ sf org list --skip-connection-status --all
```

List orgs and remove local org authorization info about non-active scratch orgs:

```
$ sf org list --clean
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--verbose`
Optional

List more information about each org.

Type: boolean

`--all`
Optional

Include expired, deleted, and unknown-status scratch orgs.

Type: boolean

`--clean`
Optional

Remove all local org authorizations for non-active scratch orgs. Use "org logout" to remove non-scratch orgs.

Type: boolean

`-p | --no-prompt`
Optional

Don't prompt for confirmation.

Type: boolean

--skip-connection-status
Optional

Skip retrieving the connection status of non-scratch orgs.

Type: boolean

Aliases for `org list`

```
force:org:list
```

`org list auth`

List authorization information about the orgs you created or logged into.

Description for `org list auth`

This command uses local authorization information that Salesforce CLI caches when you create a scratch org or log into an org. The command doesn't actually connect to the orgs to verify that they're still active. As a result, this command executes very quickly. If you want to view live information about your authorized orgs, such as their connection status, use the "org list" command.

Examples for `org list auth`

List local authorization information about your orgs:

```
sf org list auth
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

Aliases for `org list auth`

```
force:auth:list
```

```
auth:list
```

`org list limits`

Display information about limits in your org.

Description for `org list limits`

For each limit, this command returns the maximum allocation and the remaining allocation based on usage. See this topic for a description of each limit: https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/resources_limits.htm.

Examples for `org list limits`

Display limits in your default org:

```
sf org list limits
```

Display limits in the org with alias "my-scratch-org":

```
sf org list limits --target-org my-scratch-org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org list limits`

```
force:limits:api:display
```

```
limits:api:display
```

`org list metadata`

List the metadata components and properties of a specified type.

Description for `org list metadata`

Use this command to identify individual components in your manifest file or if you want a high-level view of particular metadata types in your org. For example, you can use this command to return a list of names of all the CustomObject or Layout components in your org, then use this information in a retrieve command that returns a subset of these components.

The username that you use to connect to the org must have the Modify All Data or Modify Metadata Through Metadata API Functions permission.

Examples for `org list metadata`

List the CustomObject components, and their properties, in the org with alias "my-dev-org":

```
$ sf org list metadata --metadata-type CustomObject --target-org my-dev-org
```

List the CustomObject components in your default org, write the output to the specified file, and use API version 57.0:

```
$ sf org list metadata --metadata-type CustomObject --api-version 57.0 --output-file
/path/to/outputfilename.txt
```

List the Dashboard components in your default org that are contained in the "folderSales" folder, write the output to the specified file, and use API version 57.0:

```
$ sf org list metadata --metadata-type Dashboard --folder folderSales --api-version 57.0
--output-file /path/to/outputfilename.txt
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`
Optional

API version to use; default is the most recent API version.

Override the api version used for api requests made by this command

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`-f | --output-file OUTPUT-FILE`
Optional

Pathname of the file in which to write the results.

Type: option

-m | --metadata-type METADATA-TYPE
Required

Metadata type to be retrieved, such as CustomObject; metadata type names are case-sensitive.

Type: option

--folder FOLDER
Optional

Folder associated with the component; required for components that use folders; folder names are case-sensitive.

Examples of metadata types that use folders are Dashboard, Document, EmailTemplate, and Report.

Type: option

Aliases for `org list metadata`

```
force:mdapi:listmetadata
```

`org list metadata-types`

Display details about the metadata types that are enabled for your org.

Description for `org list metadata-types`

The information includes Apex classes and triggers, custom objects, custom fields on standard objects, tab sets that define an app, and many other metadata types. Use this information to identify the syntax needed for a <name> element in a manifest file (package.xml).

The username that you use to connect to the org must have the Modify All Data or Modify Metadata Through Metadata API Functions permission.

Examples for `org list metadata-types`

Display information about all known and enabled metadata types in the org with alias "my-dev-org" using API version 57.0:

```
$ sf org list metadata-types --api-version 57.0 --target-org my-dev-org
```

Display only the metadata types that aren't yet supported by Salesforce CLI in your default org and write the results to the specified file:

```
$ sf org list metadata-types --output-file /path/to/outputfilename.txt --filter-known
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--api-version API-VERSION
Optional

API version to use; default is the most recent API version.

Override the api version used for api requests made by this command

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-f | --output-file OUTPUT-FILE
Optional

Pathname of the file in which to write the results.

Directing the output to a file makes it easier to extract relevant information for your package.xml manifest file. The default output destination is the terminal or command window console.

Type: option

Aliases for `org list metadata-types`

```
force:mdapi:describemetadata
```

`org list shape`

List all org shapes you've created.

Description for `org list shape`

The output includes the alias, username, and ID of the source org, the status of the org shape creation, and more. Use the org ID to update your scratch org configuration file so you can create a scratch org based on this org shape.

Examples for `org list shape`

List all org shapes you've created:

```
sf org list shape
```

List all org shapes in JSON format and write the output to a file:

```
sf org list shape --json > tmp/MyOrgShapeList.json
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

Aliases for `org list shape`

```
force:org:shape:list
```

`org list snapshot`

List scratch org snapshots.

Description for `org list snapshot`

You can view all the snapshots in a Dev Hub that you have access to. If you're an admin, you can see all snapshots associated with the Dev Hub org. If you're a user, you can see only your snapshots unless a Dev Hub admin gives you View All permissions.

To create a snapshot, use the "sf org create snapshot" command. To get details about a snapshot request, use "sf org get snapshot".

Examples for `org list snapshot`

List snapshots in the default Dev Hub:

```
sf org list snapshot
```

List snapshots in the Dev Hub with alias SnapshotDevHub:

```
sf org list snapshot --target-dev-hub SnapshotDevHub
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org list snapshot`

```
force:org:snapshot:list
```

`org list sobject record-counts`

Display record counts for the specified standard or custom objects.

Description for `org list sobject record-counts`

Use this command to get an approximate count of the records in standard or custom objects in your org. These record counts are the same as the counts listed in the Storage Usage page in the Setup UI. The record counts are approximate because they're calculated asynchronously and your org's storage usage isn't updated immediately. To display all available record counts, run the command without the --sobject flag.

Examples for `org list sobject record-counts`

Display all available record counts in your default org:

```
sf org list sobject record-counts
```

Display record counts for the Account, Contact, Lead, and Opportunity objects in your default org:

```
sf org list sobject record-counts --sobject Account --sobject Contact --sobject Lead
--sobject Opportunity
```

Display record counts for the Account and Lead objects for the org with alias "my-scratch-org":

```
sf org list sobject record-counts --sobject Account --sobject Lead --target-org
my-scratch-org
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-s | --sobject SOBJECT`

Optional

API name of the standard or custom object for which to display record counts.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org list sobject record-counts`

```
force:limits:recordcounts:display
```

```
limits:recordcounts:display
```

`org list users`

List all locally-authenticated users of an org.

Description for `org list users`

For scratch orgs, the list includes any users you've created with the "org create user" command; the original scratch org admin user is marked with "(A)". For other orgs, the list includes the users you used to authenticate to the org.

Examples for `org list users`

List the locally-authenticated users of your default org:

```
sf org list users
```

List the locally-authenticated users of the specified org:

```
sf org list users --target-org me@my.org
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `org list users`

```
force:user:list
```

`org login access-token`

Authorize an org using an existing Salesforce access token.

Description for `org login access-token`

By default, the command runs interactively and asks you for the access token. If you previously authorized the org, the command prompts whether you want to overwrite the local file. Specify --no-prompt to not be prompted.

To use the command in a CI/CD script, set the SF_ACCESS_TOKEN environment variable to the access token. Then run the command with the --no-prompt parameter.

Examples for `org login access-token`

Authorize an org on https://mycompany.my.salesforce.com; the command prompts you for the access token:

```
sf org login access-token --instance-url https://mycompany.my.salesforce.com
```

Authorize the org without being prompted; you must have previously set the SF_ACCESS_TOKEN environment variable to the access token:

```
sf org login access-token --instance-url https://dev-hub.my.salesforce.com --no-prompt
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-r | --instance-url INSTANCE-URL

Required

URL of the instance that the org lives on.

If you specify an --instance-url value, this value overrides the sfdcLoginUrl value in your sfdx-project.json file.

To specify a My Domain URL, use the format "https://<MyDomainName>.my.salesforce.com".

To specify a sandbox, set --instance-url to "https://<MyDomainName>--<SandboxName>.sandbox.my.salesforce.com".

Type: option

-d | --set-default-dev-hub
Optional

Set the authenticated org as the default Dev Hub.

Type: boolean

-s | --set-default
Optional

Set the authenticated org as the default that all org-related commands run against.

Type: boolean

-a | --alias ALIAS
Optional

Alias for the org.

Type: option

-p | --no-prompt
Optional

Don't prompt for confirmation.

Type: boolean

Aliases for `org login access-token`

```
force:auth:accesstoken:store
```

```
auth:accesstoken:store
```

`org login device`

Authorize an org using a device code.

Description for `org login device`

Use this command to allow a device to connect to an org.

When you run this command, it first displays an 8-digit device code and the URL for verifying the code on your org. The default instance URL is https://login.salesforce.com, so if the org you're authorizing is on a different instance, use the --instance-url. The command waits while you complete the verification. Open a browser and navigate to the displayed verification URL, enter the code, then click Connect. If you aren't already logged into your org, log in, and then you're prompted to allow the device to connect to the org. After you successfully authorize the org, you can close the browser window.

Examples for `org login device`

Authorize an org using a device code, give the org the alias TestOrg1, and set it as your default Dev Hub org:

```
sf org login device --set-default-dev-hub --alias TestOrg1
```

Authorize an org in which you've created a custom connected app with the specified client ID (consumer key):

```
sf org login device --client-id <OAuth client id>
```

Authorize a sandbox org with the specified instance URL:

```
sf org login device --instance-url
https://MyDomainName--SandboxName.sandbox.my.salesforce.com
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-i | --client-id CLIENT-ID`
Optional

OAuth client ID (also called consumer key) of your custom connected app.

Type: option

`-r | --instance-url INSTANCE-URL`
Optional

URL of the instance that the org lives on.

If you specify an --instance-url value, this value overrides the sfdcLoginUrl value in your sfdx-project.json file.

To specify a My Domain URL, use the format "https://<MyDomainName>.my.salesforce.com".

To specify a sandbox, set --instance-url to "https://<MyDomainName>--<SandboxName>.sandbox.my.salesforce.com".

Type: option

`-d | --set-default-dev-hub`
Optional

Set the authenticated org as the default Dev Hub.

Type: boolean

`-s | --set-default`
Optional

Set the authenticated org as the default that all org-related commands run against.

Type: boolean

-a | --alias ALIAS

Optional

Alias for the org.

Type: option

Aliases for `org login device`

```
force:auth:device:login
```

```
auth:device:login
```

`org login jwt`

Log in to a Salesforce org using a JSON web token (JWT).

Description for `org login jwt`

Use this command in automated environments where you can't interactively log in with a browser, such as in CI/CD scripts.

Logging into an org authorizes the CLI to run other commands that connect to that org, such as deploying or retrieving a project. You can log into many types of orgs, such as sandboxes, Dev Hubs, Env Hubs, production orgs, and scratch orgs.

Complete these steps before you run this command:

1. Create a digital certificate (also called digital signature) and the private key to sign the certificate. You can use your own key and certificate issued by a certification authority. Or use OpenSSL to create a key and a self-signed digital certificate.

2. Store the private key in a file on your computer. When you run this command, you set the --jwt-key-file flag to this file.

3. Create a custom connected app in your org using the digital certificate. Make note of the consumer key (also called client id) that's generated for you. Be sure the username of the user logging in is approved to use the connected app. When you run this command, you set the --client-id flag to the consumer key.

See https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_jwt_flow.htm for more information.

We recommend that you set an alias when you log into an org. Aliases make it easy to later reference this org when running commands that require it. If you don't set an alias, you use the username that you specified when you logged in to the org. If you run multiple commands that reference the same org, consider setting the org as your default. Use --set-default for your default scratch org or sandbox, or --set-default-dev-hub for your default Dev Hub.

Examples for `org login jwt`

Log into an org with username jdoe@example.org and on the default instance URL (https://login.salesforce.com). The private key is stored in the file /Users/jdoe/JWT/server.key and the command uses the connected app with consumer key (client id) 04580y4051234051.

```
sf org login jwt --username jdoe@example.org --jwt-key-file /Users/jdoe/JWT/server.key
--client-id 04580y4051234051
```

Set the org as the default and give it an alias:

```
sf org login jwt --username jdoe@example.org --jwt-key-file /Users/jdoe/JWT/server.key
--client-id 04580y4051234051 --alias ci-org --set-default
```

Set the org as the default Dev Hub and give it an alias:

```
sf org login jwt --username jdoe@example.org --jwt-key-file /Users/jdoe/JWT/server.key
--client-id 04580y4051234051 --alias ci-dev-hub --set-default-dev-hub
```

Log in to a sandbox using URL https://MyDomainName--SandboxName.sandbox.my.salesforce.com:

```
sf org login jwt --username jdoe@example.org --jwt-key-file /Users/jdoe/JWT/server.key
--client-id 04580y4051234051 --alias ci-org --set-default --instance-url
https://MyDomainName--SandboxName.sandbox.my.salesforce.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --username USERNAME
Required

Username of the user logging in.

Type: option

-f | --jwt-key-file JWT-KEY-FILE
Required

Path to a file containing the private key.

Type: option

-i | --client-id CLIENT-ID
Required

OAuth client ID (also called consumer key) of your custom connected app.

Type: option

-r | --instance-url INSTANCE-URL
Optional

URL of the instance that the org lives on.

If you specify an --instance-url value, this value overrides the sfdcLoginUrl value in your sfdx-project.json file.

To specify a My Domain URL, use the format "https://<MyDomainName>.my.salesforce.com".

To specify a sandbox, set --instance-url to "https://<MyDomainName>--<SandboxName>.sandbox.my.salesforce.com".

Type: option

-d | --set-default-dev-hub
Optional

Set the authenticated org as the default Dev Hub.

Type: boolean

-s | --set-default
Optional

Set the authenticated org as the default that all org-related commands run against.

Type: boolean

-a | --alias ALIAS
Optional

Alias for the org.

Type: option

Aliases for `org login jwt`

```
force:auth:jwt:grant
```

```
auth:jwt:grant
```

`org login sfdx-url`

Authorize an org using a Salesforce DX authorization URL stored in a file or through standard input (stdin).

Description for `org login sfdx-url`

You use the Salesforce DX (SFDX) authorization URL to authorize Salesforce CLI to connect to a target org. The URL contains the required data to accomplish the authorization, such as the client ID, client secret, and instance URL. You must specify the SFDX authorization URL in this format: "force://<clientId>:<clientSecret>:<refreshToken>@<instanceUrl>". Replace <clientId>, <clientSecret>, <refreshToken>, and <instanceUrl> with the values specific to your target org. For <instanceUrl>, don't include a protocol (such as "https://"). Note that although the SFDX authorization URL starts with "force://", it has nothing to do with the actual authorization. Salesforce CLI always communicates with your org using HTTPS.

To see an example of an SFDX authorization URL, run "org display --verbose" on an org.

You have three options when creating the authorization file. The easiest option is to redirect the output of the "sf org display --verbose --json" command into a file. For example, using an org with alias my-org that you've already authorized:

```
    $ sf org display --target-org my-org --verbose --json > authFile.json
```

The resulting JSON file contains the URL in the "sfdxAuthUrl" property of the "result" object. You can then reference the file when running this command:

```
    $ sf org login sfdx-url --sfdx-url-file authFile.json
```

NOTE: The "sf org display --verbose" command displays the refresh token only for orgs authorized with the web server flow, and not the JWT bearer flow.

You can also create a JSON file that has a top-level property named sfdxAuthUrl whose value is the authorization URL. Finally, you can create a normal text file that includes just the URL and nothing else.

Alternatively, you can pipe the SFDX authorization URL through standard input by specifying the --sfdx-url-stdin flag.

Examples for `org login sfdx-url`

Authorize an org using the SFDX authorization URL in the files/authFile.json file:

```
sf org login sfdx-url --sfdx-url-file files/authFile.json
```

Similar to previous example, but set the org as your default and give it an alias MyDefaultOrg:

```
sf org login sfdx-url --sfdx-url-file files/authFile.json --set-default --alias MyDefaultOrg
```

Pipe the SFDX authorization URL from stdin:

```
$ echo url | sf org login sfdx-url --sfdx-url-stdin
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-f | --sfdx-url-file SFDX-URL-FILE`
Optional

Path to a file that contains the Salesforce DX authorization URL.

Type: option

`-u | --sfdx-url-stdin SFDX-URL-STDIN`
Optional

Pipe the Salesforce DX authorization URL through standard input (stdin).

Type: option

`-d | --set-default-dev-hub`
Optional

Set the authenticated org as the default Dev Hub.

Type: boolean

`-s | --set-default`
Optional

Set the authenticated org as the default that all org-related commands run against.

Type: boolean

`-a | --alias ALIAS`
Optional

Alias for the org.

Type: option

Aliases for `org login sfdx-url`

```
force:auth:sfdxurl:store
```

```
auth:sfdxurl:store
```

`org login web`

Log in to a Salesforce org using the web server flow.

Description for `org login web`

Opens a Salesforce instance URL in a web browser so you can enter your credentials and log in to your org. After you log in, you can close the browser window.

Logging into an org authorizes the CLI to run other commands that connect to that org, such as deploying or retrieving metadata. You can log into many types of orgs, such as sandboxes, Dev Hubs, Env Hubs, production orgs, and scratch orgs.

We recommend that you set an alias when you log into an org. Aliases make it easy to later reference this org when running commands that require it. If you don't set an alias, you use the username that you specified when you logged in to the org. If you run multiple commands that reference the same org, consider setting the org as your default. Use --set-default for your default scratch org or sandbox, or --set-default-dev-hub for your default Dev Hub.

By default, this command uses the global out-of-the-box connected app in your org. If you need more security or control, such as setting the refresh token timeout or specifying IP ranges, create your own connected app using a digital certificate. Make note of the consumer key (also called cliend id) that's generated for you. Then specify the consumer key with the --client-id flag.

Examples for `org login web`

Run the command with no flags to open the default Salesforce login page (https://login.salesforce.com):

```
sf org login web
```

Log in to your Dev Hub, set it as your default Dev Hub, and set an alias that you reference later when you create a scratch org:

```
sf org login web --set-default-dev-hub --alias dev-hub
```

Log in to a sandbox and set it as your default org:

```
sf org login web --instance-url https://MyDomainName--SandboxName.sandbox.my.salesforce.com
 --set-default
```

Use --browser to specify a specific browser, such as Google Chrome:

```
sf org login web --instance-url https://MyDomainName--SandboxName.sandbox.my.salesforce.com
 --set-default --browser chrome
```

Use your own connected app by specifying its consumer key (also called client ID):

```
sf org login web --instance-url https://MyDomainName--SandboxName.sandbox.my.salesforce.com
 --set-default --browser chrome --client-id 04580y4051234051
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-b | --browser BROWSER
Optional

Browser in which to open the org.

If you don't specify --browser, the command uses your default browser. The exact names of the browser applications differ depending on the operating system you're on; check your documentation for details.

Type: option

Permissible values are: chrome, edge, firefox

-i | --client-id CLIENT-ID
Optional

OAuth client ID (also called consumer key) of your custom connected app.

Type: option

-r | --instance-url INSTANCE-URL
Optional

URL of the instance that the org lives on.

If you specify an --instance-url value, this value overrides the sfdcLoginUrl value in your sfdx-project.json file.

To specify a My Domain URL, use the format "https://<MyDomainName>.my.salesforce.com".

To specify a sandbox, set --instance-url to "https://<MyDomainName>--<SandboxName>.sandbox.my.salesforce.com".

Type: option

-d | --set-default-dev-hub
Optional

Set the authenticated org as the default Dev Hub.

Type: boolean

-s | --set-default
Optional

Set the authenticated org as the default that all org-related commands run against.

Type: boolean

-a | --alias ALIAS
Optional

Alias for the org.

Type: option

Aliases for `org login web`

```
force:auth:web:login
```

```
auth:web:login
```

`org logout`

Log out of a Salesforce org.

Description for `org logout`

If you run this command with no flags and no default org set in your config or environment, it first displays a list of orgs you've created or logged into, with none of the orgs selected. Use the arrow keys to scroll through the list and the space bar to select the orgs you want to log out of. Press Enter when you're done; the command asks for a final confirmation before logging out of the selected orgs.

The process is similar if you specify --all, except that in the initial list of orgs, they're all selected. Use --target-org to logout of a specific org. In both these cases by default, you must still confirm that you want to log out. Use --no-prompt to never be asked for confirmation when also using --all or --target-org.

Be careful! If you log out of a scratch org without having access to its password, you can't access the scratch org again, either through the CLI or the Salesforce UI.

Examples for `org logout`

Interactively select the orgs to log out of:

```
sf org logout
```

Log out of the org with username me@my.org:

```
sf org logout --target-org me@my.org
```

Log out of all orgs after confirmation:

```
sf org logout --all
```

Logout of the org with alias my-scratch and don't prompt for confirmation:

```
sf org logout --target-org my-scratch --no-prompt
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org.

Type: option

-a | --all
Optional

Include all authenticated orgs.

All orgs includes Dev Hubs, sandboxes, DE orgs, and expired, deleted, and unknown-status scratch orgs.

Type: boolean

-p | --no-prompt
Optional

Don't prompt for confirmation.

Type: boolean

Aliases for `org logout`

```
force:auth:logout
```

```
auth:logout
```

`org open`

Open your default scratch org, or another specified org, in a browser.

Description for `org open`

To open a specific page, specify the portion of the URL after "https://mydomain.my.salesforce.com" as the value for the --path flag. For example, specify "--path lightning" to open Lightning Experience, or specify "--path /apex/YourPage" to open a Visualforce page.

Use the --source-file flag to open ApexPage, FlexiPage, Flow, or Agent metadata from your local project in the associated Builder within the Org.

To generate a URL but not launch it in your browser, specify --url-only.

To open in a specific browser, use the --browser flag. Supported browsers are "chrome", "edge", and "firefox". If you don't specify --browser, the org opens in your default browser.

Examples for `org open`

Open your default org in your default browser:

```
$ sf org open
```

Open your default org in an incognito window of your default browser:

```
$ sf org open --private
```

Open the org with alias MyTestOrg1 in the Firefox browser:

```
$ sf org open --target-org MyTestOrg1 --browser firefox
```

Display the navigation URL for the Lightning Experience page for your default org, but don't open the page in a browser:

```
$ sf org open --url-only --path lightning
```

Open a local Lightning page in your default org's Lightning App Builder:

```
$ sf org open --source-file force-app/main/default/flexipages/Hello.flexipage-meta.xml
```

Open a local Flow in Flow Builder:

```
$ sf org open --source-file force-app/main/default/flows/Hello.flow-meta.xml
```

Open local Agent metadata (Bot) in Agent Builder:

```
$ sf org open --source-file
force-app/main/default/bots/Coral_Cloud_Agent/Coral_Cloud_Agent.bot-meta.xml
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

--private
Optional

Open the org in the default browser using private (incognito) mode.

Type: boolean

-b | --browser BROWSER
Optional

Browser where the org opens.

Type: option

Permissible values are: chrome, edge, firefox

-p | --path PATH
Optional

Navigation URL path to open a specific page.

Type: option

-r | --url-only
Optional

Display navigation URL, but don't launch browser.

Type: boolean

-f | --source-file SOURCE-FILE
Optional

Path to ApexPage, FlexiPage, Flow, or Agent metadata to open in the associated Builder.

Type: option

Aliases for `org open`

```
force:org:open
```

```
force:source:open
```

`org open agent` (Beta)

Open an agent in your org's Agent Builder UI in a browser.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `org open agent`

Use the --name flag to open an agent using its API name in the Agent Builder UI of your org. To find the agent's API name, go to Setup in your org and navigate to the agent's details page.

To generate the URL but not launch it in your browser, specify --url-only.

To open Agent Builder in a specific browser, use the --browser flag. Supported browsers are "chrome", "edge", and "firefox". If you don't specify --browser, the org opens in your default browser.

Examples for `org open agent`

Open the agent with API name Coral_Cloud_Agent in your default org using your default browser:

```
$ sf org open agent --name Coral_Cloud_Agent
```

Open the agent in an incognito window of your default browser:

```
$ sf org open agent --private --name Coral_Cloud_Agent:
```

Open the agent in an org with alias MyTestOrg1 using the Firefox browser:

```
$ sf org open agent --target-org MyTestOrg1 --browser firefox --name Coral_Cloud_Agent
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-n | --name NAME
Required

API name, also known as developer name, of the agent you want to open in the org's Agent Builder UI.

Type: option

--private
Optional

Open the org in the default browser using private (incognito) mode.

Type: boolean

-b | --browser BROWSER
Optional

Browser where the org opens.

Type: option

Permissible values are: chrome, edge, firefox

-r | --url-only
Optional

Display navigation URL, but don’t launch browser.

Type: boolean

org refresh sandbox

Refresh a sandbox org using the sandbox name.

Description for `org refresh sandbox`

Refreshing a sandbox copies the metadata, and optionally data, from your source org to the refreshed sandbox org. You can optionally specify a definition file if you want to change the configuration of the refreshed sandbox, such as its license type or template ID. You can also use the --source-id or --source-sandbox-name flags to change the refreshed sandbox org's original source org to a new org; in this case, the refreshed sandbox org's metadata is updated with the new source org's metadata.

You're not allowed to change the sandbox name when you refresh it with this command. If you want to change the sandbox name, first delete it with the "org delete sandbox" command. And then recreate it with the "org create sandbox" command and give it a new name.

Examples for `org refresh sandbox`

Refresh the sandbox named "devSbx1". The production org that contains the sandbox license has the alias "prodOrg".

```
sf org refresh sandbox --name devSbx1 --target-org prodOrg
```

Refresh the sandbox named "devSbx2", and override the configuration of the refreshed sandbox with the properties in the specified defintion file. The default target org is the production org, so you don't need to specify the `--target-org` flag in this case.

```
sf org refresh sandbox --name devSbx2 --definition-file devSbx2-config.json
```

Refresh the sandbox using the name defined in the definition file. The production org that contains the sandbox license has the alias "prodOrg".

```
sf org refresh sandbox --definition-file devSbx3-config.json --target-org prodOrg
```

Refresh the sandbox named "devSbx2" by changing its original source org to be a sandbox called "devSbx3":

```
sf org refresh sandbox --name devSbx2 --source-sandbox-name devSbx3 --target-org prodOrg
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--no-auto-activate`
Optional

Disable auto-activation of the sandbox after a successful refresh.

By default, a sandbox auto-activates after a refresh. Use this flag to control sandbox activation manually.

Type: boolean

`-w | --wait WAIT`
Optional

Number of minutes to poll for sandbox refresh status.

If the command continues to run after the wait period, the CLI returns control of the terminal to you and displays the "sf org resume sandbox" command for you run to check the status of the refresh. The displayed command includes the job ID for the running sandbox refresh.

Type: option

Default value: 30 minutes

-i | --poll-interval POLL-INTERVAL

Optional

Number of seconds to wait between status polling requests.

Type: option

Default value: 30 seconds

--source-sandbox-name SOURCE-SANDBOX-NAME

Optional

Name of the sandbox org that becomes the new source org for the refreshed sandbox.

The value of --source-sandbox-name must be an existing sandbox. The new source sandbox, and the refreshed sandbox specified with the --name flag, must both be associated with the production org (--target-org) that contains the sandbox licenses.

You can specify either --source-sandbox-name or --source-id when refreshing an existing sandbox, but not both.

Type: option

--source-id SOURCE-ID

Optional

ID of the sandbox org that becomes the new source org for the refreshed sandbox.

The value of --source-id must be an existing sandbox. The new source sandbox, and the refreshed sandbox specified with the --name flag, must both be associated with the production org (--target-org) that contains the sandbox licenses.

You can specify either --source-id or --source-sandbox-name when refreshing an existing sandbox, but not both.

Type: option

--async

Optional

Request the sandbox refresh, but don't wait for it to complete.

The command immediately displays the job ID and returns control of the terminal to you. This way, you can continue to use the CLI. To check the status of the sandbox refresh, run "sf org resume sandbox".

Type: boolean

-n | --name NAME

Optional

Name of the existing sandbox org in your production org that you want to refresh.

Type: option

-f | --definition-file DEFINITION-FILE

Optional

Path to a sandbox definition file for overriding its configuration when you refresh it.

The sandbox definition file is a blueprint for the sandbox; use the file to change the sandbox configuration during a refresh. If you don't want to change the sandbox configuration when you refresh it, then simply use the --name flag to specify the sandbox and don't use this flag. See

<https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_sandbox_definition.htm> for all the options you can specify in the definition file.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the production org that contains the sandbox license.

Type: option

--no-prompt
Optional

Don't prompt for confirmation about the sandbox refresh.

Type: boolean

org resume sandbox

Check the status of a sandbox creation, and log in to it if it's ready.

Description for org resume sandbox

Sandbox creation can take a long time. If the original "sf org create sandbox" command either times out, or you specified the --async flag, the command displays a job ID. Use this job ID to check whether the sandbox creation is complete, and if it is, the command then logs into it.

You can also use the sandbox name to check the status or the --use-most-recent flag to use the job ID of the most recent sandbox creation.

Examples for org resume sandbox

Check the status of a sandbox creation using its name and specify a production org with alias "prodOrg":

```
sf org resume sandbox --name mysandbox --target-org prodOrg
```

Check the status using the job ID:

```
sf org resume sandbox --job-id 0GRxxxxxxxx
```

Check the status of the most recent sandbox create request:

```
sf org resume sandbox --use-most-recent
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the sandbox org to be ready.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To resume checking the sandbox creation, rerun this command.

Type: option

-n | --name NAME
Optional

Name of the sandbox org.

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the incomplete sandbox creation that you want to check the status of.

The job ID is valid for 24 hours after you start the sandbox creation.

Type: option

-l | --use-most-recent
Optional

Use the most recent sandbox create request.

Type: boolean

-o | --target-org TARGET-ORG
Optional

Username or alias of the production org that contains the sandbox license.

When it creates the sandbox org, Salesforce copies the metadata, and optionally data, from your production org to the new sandbox org.

Type: option

Aliases for `org resume sandbox`

```
env:resume:sandbox
```

`org resume scratch`

Resume the creation of an incomplete scratch org.

Description for `org resume scratch`

When the original "sf org create scratch" command either times out or is run with the --async flag, it displays a job ID.

Run this command by either passing it a job ID or using the --use-most-recent flag to specify the most recent incomplete scratch org.

Examples for `org resume scratch`

Resume a scratch org create with a job ID:

```
sf org resume scratch --job-id 2SR3u0000008fBDGAY
```

Resume your most recent incomplete scratch org:

```
sf org resume scratch --use-most-recent
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-i | --job-id JOB-ID`
Optional

Job ID of the incomplete scratch org create that you want to resume.

The job ID is the same as the record ID of the incomplete scratch org in the ScratchOrgInfo object of the Dev Hub.

The job ID is valid for 24 hours after you start the scratch org creation.

Type: option

`-r | --use-most-recent`
Optional

Use the job ID of the most recent incomplete scratch org.

Type: boolean

`-w | --wait WAIT`
Optional

Number of minutes to wait for the scratch org to be ready.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To resume checking the scratch creation, rerun this command.

Type: option

Aliases for `org resume scratch`

```
env:resume:scratch
```

`package` Commands

Commands to develop and install unlocked packages and managed 2GP packages.

package create

Create a package.

Description for `package create`

First, use this command to create a package. Then create a package version.

If you don't have a namespace defined in your sfdx-project.json file, use --no-namespace.

Your --name value must be unique within your namespace.

Run 'sf package list to list all packages in the Dev Hub org.

Examples for `package create`

Create an unlocked package from the files in the "force-app" directory; uses your default Dev Hub org:

```
sf package create --name MyUnlockedPackage --package-type Unlocked --path force-app
```

Create a managed packaged from the "force-app" directory files, give the package a description, and use the specified Dev Hub org:

```
sf package create --name MyManagedPackage --description "Your Package Descripton"
--package-type Managed --path force-app --target-dev-hub devhub@example.com
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-n | --name NAME`
Required

Name of the package to create.

Type: option

`-t | --package-type PACKAGE-TYPE`
Required

Type of package.

The options for package type are Managed and Unlocked (Managed=DeveloperManagedSubscriberManaged, Unlocked=DeveloperControlledSubscriberEditable). These options determine upgrade and editability rules.

Type: option

Permissible values are: Managed, Unlocked

-d | --description DESCRIPTION
Optional

Description of the package.

Type: option

-e | --no-namespace
Optional

Create the package with no namespace; available only for unlocked packages.

This flag is useful when you're migrating an existing org to packages. But use a namespaced package for new metadata.

Type: boolean

-r | --path PATH
Required

Path to directory that contains the contents of the package.

Type: option

--org-dependent
Optional

Depends on unpackaged metadata in the installation org; applies to unlocked packages only.

Use Source Tracking in Sandboxes to develop your org-dependent unlocked package. For more information, see "Create Org-Dependent Unlocked Packages" in the Salesforce DX Developer Guide.

Type: boolean

-o | --error-notification-username ERROR-NOTIFICATION-USERNAME
Optional

Active Dev Hub user designated to receive email notifications for package errors.

Email notifications include information about unhandled Apex exceptions, and install, upgrade, or uninstall failures associated with your package.

Type: option

Aliases for `package create`

```
force:package:create
```

`package delete`

Delete a package.

Description for `package delete`

Specify the ID or alias of the package you want to delete.

Delete unlocked and second-generation managed packages. Before you delete a package, first delete all associated package versions.

Examples for `package delete`

Delete a package using its alias from your default Dev Hub org:

```
sf package delete --package "Your Package Alias"
```

Delete a package using its ID from the specified Dev Hub org:

```
sf package delete --package 0Ho... --target-dev-hub devhub@example.com
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-n | --no-prompt`
Optional

Don't prompt before deleting the package.

Type: boolean

`-p | --package PACKAGE`
Required

ID (starts with 0Ho) or alias of the package to delete.

Type: option

Aliases for `package delete`

```
force:package:delete
```

`package install`

Install or upgrade a version of a package in the target org.

Description for `package install`

To install or upgrade a package, specify a specific version of the package using the 04t package ID. The package and the version you specified installs in your default target org unless you supply the username for a different target org.

When upgrading an unlocked package, include the --upgrade-type value to specify whether any removed components are deprecated or deleted. To delete components that can be safely deleted and deprecate the others, specify "--upgrade-type Mixed" (the default). To deprecate all removed components, specify "--upgrade-type DeprecateOnly". To delete all removed components, except for custom objects and custom fields, that don't have dependencies, specify "--upgrade-type Delete". (Note: This option can result in the loss of data that is associated with the deleted components.)

Examples for `package install`

Install or upgrade a package version with the specified ID in the org with username "me@example.com":

```
sf package install --package 04t... --target-org me@example.com
```

Install or upgrade a package version with the specified alias into your default org:

```
sf package install --package awesome_package_alias
```

Install or upgrade a package version with an alias that includes spaces into your default org:

```
sf package install --package "Awesome Package Alias"
```

Upgrade an unlocked package version with the specified ID and deprecate all removed components:

```
sf package install --package 04t... --upgrade-type DeprecateOnly
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-w | --wait WAIT`
Optional

Number of minutes to wait for installation status.

Type: option

Default value: 0 minutes

-k | --installation-key INSTALLATION-KEY
Optional

Installation key for key-protected package (default: null).

Type: option

-b | --publish-wait PUBLISH-WAIT
Optional

Maximum number of minutes to wait for the Subscriber Package Version ID to become available in the target org before canceling the install request.

Type: option

Default value: 0 minutes

-r | --no-prompt
Optional

Don't prompt for confirmation.

Allows the following without an explicit confirmation response: 1) Remote Site Settings and Content Security Policy websites to send or receive data, and 2) --upgrade-type Delete to proceed.

Type: boolean

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package version to install.

Type: option

-a | --apex-compile APEX-COMPILE
Optional

Compile all Apex in the org and package, or only Apex in the package; unlocked packages only.

Applies to unlocked packages only. Specifies whether to compile all Apex in the org and package, or only the Apex in the package.

For package installs into production orgs, or any org that has Apex Compile on Deploy enabled, the platform compiles all Apex in the org after the package install or upgrade operation completes.

This approach assures that package installs and upgrades don’t impact the performance of an org, and is done even if --apex-compile package is specified.

Type: option

Permissible values are: all, package

Default value: all

-s | --security-type SECURITY-TYPE
Optional

Security access type for the installed package. Available options are AdminsOnly and AllUsers.

Type: option

Permissible values are: AllUsers, AdminsOnly

Default value: AdminsOnly

-t | --upgrade-type UPGRADE-TYPE

Optional

Upgrade type for the package installation; available only for unlocked packages.

For unlocked package upgrades, set this flag to one of these values:

- DeprecateOnly: Mark all removed components as deprecated.

- Mixed: Delete removed components, except for custom objects and custom fields, that don't have dependencies.

- Delete: Delete all removed components that can be safely deleted, and deprecate the other components.

Type: option

Permissible values are: DeprecateOnly, Mixed, Delete

Default value: Mixed

Aliases for `package install`

```
force:package:install
```

`package install report`

Retrieve the status of a package installation request.

Examples for `package install report`

Retrieve the status of a package installation request with the specified ID on your default org:

```
sf package install report --request-id 0Hf...
```

Similar to previous example, except use the org with username me@example.com:

```
sf package install report --request-id 0Hf... --target-org me@example.com
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --request-id REQUEST-ID
Required

ID of the package install request you want to check; starts with 0Hf.

Type: option

Aliases for `package install report`

```
force:package:install:report
```

package installed list

List the org's installed packages.

Examples for `package installed list`

List the installed packages in your default org:

```
sf package installed list
```

List the installed packages in the org with username me@example.com:

```
sf package installed list --target-org me@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `package installed list`

```
force:package:installed:list
```

`package list`

List all packages in the Dev Hub org.

Description for `package list`

Description

Examples for `package list`

List all packages in the specified Dev Hub org:

```
sf package list --target-dev-hub devhub@example.com
```

List all packages details in the specified Dev Hub org, and show extended details about each package:

```
sf package list --target-dev-hub devhub@example.com --verbose
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`--verbose`
Optional

Display extended package detail.

Type: boolean

Aliases for `package list`

```
force:package:list
```

`package uninstall`

Uninstall a second-generation package from the target org.

Description for `package uninstall`

Specify the package ID for a second-generation package.

To list the org's installed packages, run "sf package installed list".

To uninstall a first-generation package, from Setup, enter Installed Packages in the Quick Find box, then select Installed Packages.

Examples for `package uninstall`

Uninstall a package with specified ID from an org with username me@example.com:

```
sf package uninstall --package 04t... --target-org me@example.com
```

Uninstall a package with the specified alias from your default org:

```
sf package uninstall --package undesirable_package_alias
```

Uninstall a package with an alias that contains spaces from your default org:

```
sf package uninstall --package "Undesirable Package Alias"
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`

Optional

Override the api version used for api requests made by this command

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for uninstall status.

Type: option

Default value: 0 minutes

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package version to uninstall.

Type: option

Aliases for `package uninstall`

```
force:package:uninstall
```

`package uninstall report`

Retrieve the status of a package uninstall request.

Examples for `package uninstall report`

Retrieve the status of a package uninstall in your default org using the specified request ID:

```
sf package uninstall report --request-id 06y...
```

Similar to previous example, but use the org with username me@example.com:

```
sf package uninstall report --request-id 06y... --target-org me@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --request-id REQUEST-ID
Required

ID of the package uninstall request you want to check; starts with 06y.

Type: option

Aliases for `package uninstall report`

```
force:package:uninstall:report
```

`package update`

Update package details.

Description for `package update`

Specify a new value for each option you want to update.

Run "sf package list" to list all packages in the Dev Hub org.

Examples for `package update`

Update the name of the package with the specified alias; uses your default Dev Hub org:

```
sf package update --package "Your Package Alias" --name "New Package Name"
```

Update the description of the package with the specified ID; uses the specified Dev Hub org:

```
sf package update --package 0Ho... --description "New Package Description" --target-dev-hub
 devhub@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-p | --package PACKAGE
Required

ID (starts with 0Ho) or alias of the package to update.

Type: option

-n | --name NAME
Optional

New name of the package.

Type: option

-d | --description DESCRIPTION
Optional

New description of the package.

Type: option

-o | --error-notification-username ERROR-NOTIFICATION-USERNAME
Optional

Active Dev Hub user designated to receive email notifications for package errors.

Email notifications include information about unhandled Apex exceptions, and install, upgrade, or uninstall failures associated with your package.

Type: option

--enable-app-analytics
Optional

Enable AppExchange App Analytics usage data collection on this managed package and its components.

Type: boolean

Aliases for `package update`

```
force:package:update
```

`package version create`

Create a package version in the Dev Hub org.

Description for `package version create`

The package version is based on the package contents in the specified directory.

To retrieve details about a package version create request, including status and package version ID (04t), run "sf package version create report -i 08c...".

We recommend that you specify the --installation-key parameter to protect the contents of your package and to prevent unauthorized installation of your package.

To list package version creation requests in the org, run "sf package version create list".

To promote a package version to released, you must use the --code-coverage parameter. The package must also meet the code coverage requirements. This requirement applies to both managed and unlocked packages.

We don't calculate code coverage for org-dependent unlocked packages, or for package versions that specify --skip-validation.

Examples for `package version create`

Create a package version from the contents of the "common" directory and give it an installation key of "password123"; uses your default Dev Hub org:

```
sf package version create --path common --installation-key password123
```

Create a package version from a package with the specified alias; uses the Dev Hub org with username devhub@example.com:

```
sf package version create --package "Your Package Alias" --installation-key password123
--target-dev-hub devhub@example.com
```

Create a package version from a package with the specified ID:

```
sf package version create --package 0Ho... --installation-key password123
```

Create a package version and skip the validation step:

```
sf package version create --path common --installation-key password123 --skip-validation
```

Create a package version and perform package validations asynchronously:

```
sf package version create --path common --installation-key password123 --async-validation
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

-b | --branch BRANCH
Optional

Name of the branch in your source control system that the package version is based on.

Type: option

-c | --code-coverage
Optional

Calculate and store the code coverage percentage by running the packaged Apex tests included in this package version.

Before you can promote and release a managed or unlocked package version, the Apex code must meet a minimum 75% code coverage requirement. We don't calculate code coverage for org-dependent unlocked packages or for package versions that specify --skip-validation.

Type: boolean

-f | --definition-file DEFINITION-FILE
Optional

Path to a definition file similar to scratch org definition file that contains the list of features and org preferences that the metadata of the package version depends on.

For a patch version, the features specified in this file are ignored, and instead the features specified for the ancestor version are used.

Type: option

-k | --installation-key INSTALLATION-KEY
Optional

Installation key for key-protected package. (either --installation-key or --installation-key-bypass is required)

Type: option

-x | --installation-key-bypass
Optional

Bypass the installation key requirement. (either --installation-key or --installation-key-bypass is required)

If you bypass this requirement, anyone can install your package.

Type: boolean

-p | --package PACKAGE
Optional

ID (starts with 0Ho) or alias of the package to create a version of.

Type: option

-d | --path PATH
Optional

Path to the directory that contains the contents of the package.

Type: option

--post-install-script POST-INSTALL-SCRIPT
Optional

Name of the post-install script; applies to managed packages only.

The post-install script is an Apex class within this package that is run in the installing org after installations or upgrades of this package version.

Type: option

--post-install-url POST-INSTALL-URL
Optional

Post-install instructions URL.

The contents of the post-installation instructions URL are displayed in the UI after installation of the package version.

Type: option

--releasenotes-url RELEASENOTES-URL
Optional

Release notes URL.

This link is displayed in the package installation UI to provide release notes for this package version to subscribers.

Type: option

--skip-ancestor-check
Optional

Overrides ancestry requirements, which allows you to specify a package ancestor that isn't the highest released package version.

Type: boolean

--skip-validation
Optional

Skip validation during package version creation; you can't promote unvalidated package versions.

Skips validation of dependencies, package ancestors, and metadata during package version creation. Skipping validation reduces the time it takes to create a new package version, but you can promote only validated package versions. Skipping validation can suppress important errors that can surface at a later stage. You can specify skip validation or code coverage, but not both. Code coverage is calculated during validation.

Type: boolean

--async-validation
Optional

Return a new package version before completing package validations.

Specifying async validation returns the package version earlier in the process, allowing you to install and test the new version right away. If your development team is using continuous integration (CI) scripts, async validation can reduce your overall CI run time.

Type: boolean

-t | --tag TAG
Optional

Package version's tag.

Type: option

--uninstall-script UNINSTALL-SCRIPT
Optional

Uninstall script name; applies to managed packages only.

The uninstall script is an Apex class within this package that is run in the installing org after uninstallations of this package.

Type: option

-e | --version-description VERSION-DESCRIPTION
Optional

Description of the package version to be created; overrides the sfdx-project.json value.

Type: option

-a | --version-name VERSION-NAME
Optional

Name of the package version to be created; overrides the sfdx-project.json value.

Type: option

-n | --version-number VERSION-NUMBER
Optional

Version number of the package version to be created; overrides the sfdx-project.json value.

For information about the format of the version number, see https://developer.salesforce.com/docs/atlas.en-us.pkg2_dev.meta/pkg2_dev/sfdx_dev2gp_config_file.htm.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the package version to be created.

Type: option

Default value: 0 minutes

--language LANGUAGE
Optional

Language for the package.

Specify the language using a language code listed under "Supported Languages" in Salesforce Help. If no language is specified, the language defaults to the language of the Dev Hub user who created the package.

Type: option

--verbose
Optional

Display verbose command output.

Display verbose command output. When polling for the status of the creation, this will output status and timeout data on a separate line for each poll request, which is useful in CI systems where timeouts can occur with long periods of no output from commands.

Type: boolean

Aliases for `package version create`

```
force:package:version:create
```

`package version create list`

List package version creation requests.

Description for `package version create list`

Shows the details of each request to create a package version in the Dev Hub org.

All filter parameters are applied using the AND logical operator (not OR).

To get information about a specific request, run "sf package version create report" and supply the request ID.

Examples for `package version create list`

List all package version creation requests in your default Dev Hub org:

```
sf package version create list
```

List package version creation requests from the last 3 days in the Dev Hub org with username devhub@example.com:

```
sf package version create list --created-last-days 3 --target-dev-hub
```

List package version creation requests with status Error:

```
sf package version create list --status Error
```

List package version creation requests with status InProgress:

```
sf package version create list --status InProgress
```

List package version creation requests with status Success that were created today:

```
sf package version create list --created-last-days 0 --status Success
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-c | --created-last-days CREATED-LAST-DAYS`
Optional

Number of days since the request was created, starting at 00:00:00 of first day to now. Use 0 for today.

Type: option

`-s | --status STATUS`
Optional

Status of the version creation request, used to filter the list.

Type: option

Permissible values are: Queued, InProgress, Success, Error

--show-conversions-only
Optional

Filter the list output to display only converted package version.

Type: boolean

--verbose
Optional

Displays additional information at a slight performance cost, such as the version name and number for each package version create request.

Type: boolean

Aliases for `package version create list`

```
force:package:version:create:list
```

`package version create report`

Retrieve details about a package version creation request.

Description for `package version create report`

Specify the request ID for which you want to view details. If applicable, the command displays errors related to the request.

To show all requests in the org, run "sf package version create list".

Examples for `package version create report`

Retrieve details about the package version creation request with the specified ID; uses your default Dev Hub org:

```
sf package version create report --package-create-request-id 08c...
```

Retrieve details about the specified package version creation request in the Dev Hub org with username devhub@example.com:

```
sf package version create report --package-create-request-id 08c... --target-dev-hub
devhub@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --package-create-request-id PACKAGE-CREATE-REQUEST-ID
Required

ID (starts with 08c) of the package version creation request you want to display.

Type: option

Aliases for `package version create report`

```
force:package:version:create:report
```

`package version delete`

Delete a package version.

Description for `package version delete`

Specify the ID or alias of the package version you want to delete. In second-generation managed packaging, only beta package versions can be deleted. Before deleting a package version, review the considerations outlined in https://developer.salesforce.com/docs/atlas.en-us.pkg2_dev.meta/pkg2_dev/sfdx_dev_dev2gp_package_deletion.htm.

Examples for `package version delete`

Delete a package version with the specified alias using your default Dev Hub org:

```
sf package version delete --package "Your Package Alias"
```

Delete a package version with the specified ID using the Dev Hub org with username "devhub@example.com":

```
sf package version delete --package 04t... --target-org devhub@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-n | --no-prompt
Optional

Don't prompt before deleting the package version.

Type: boolean

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package version to delete.

Type: option

Aliases for `package version delete`

```
force:package:version:delete
```

`package version displayancestry`

Display the ancestry tree for a 2GP managed package version.

Examples for `package version displayancestry`

Display the ancestry tree for a package version with the specified alias, using your default Dev Hub org:

```
sf package version displayancestry --package package_version_alias
```

Similar to previous example, but display the output in DOT code:

```
sf package version displayancestry --package package_version_alias --dot-code
```

Display the ancestry tree for a package with the specified ID, using the Dev Hub org with username devhub@example.com:

```
sf package version displayancestry --package OHo... --target-dev-hub devhub@example.com
```

Display the ancestry tree of a package version with the specified ID, using your default Dev Hub org:

```
sf package version displayancestry --package 04t...
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-p | --package PACKAGE
Required

ID or alias of the package (starts with 0Ho) or package version (starts with 04t) to display ancestry for.

If you specify a package ID (starts with 0Ho) or alias, the ancestor tree for every package version associated with the package ID is displayed. If you specify a package version (starts with 04t) or alias, the ancestry tree of the specified package version is displayed.

Type: option

--dot-code
Optional

Display the ancestry tree in DOT code.

You can use the DOT code output in graph visualization software to create tree visualizations.

Type: boolean

--verbose
Optional

Display both the package version ID (starts with 04t) and the version number (major.minor.patch.build) in the ancestry tree.

Type: boolean

Aliases for `package version displayancestry`

```
force:package:version:displayancestry
```

`package version list`

List all package versions in the Dev Hub org.

Description for `package version list`

Description

Examples for `package version list`

List package versions in your default Dev Hub org that were created in the last 3 days; show only the released versions and order the list using the PatchVersion field. Display extended details about each package version:

```
sf package version list --verbose --created-last-days 3 --released --order-by PatchVersion
```

List the released package versions for the two specified packages that were modified today; use the Dev Hub org with username devhub@example.com:

```
sf package version list --packages 0Ho000000000000,0Ho000000000001 --released
--modified-last-days 0 --target-dev-hub devhub@example.com
```

List all released package versions in your default Dev Hub org:

```
sf package version list --released
```

List package versions that were modified today in your default Dev Hub org; show limited details about each one:

```
sf package version list --concise --modified-last-days 0
```

List package versions that are based on the "featureA" branch in your source control system that were modified today in your default Dev Hub org; show limited details about each one:

```
sf package version list --concise --modified-last-days 0 --branch featureA
```

List released package versions that were created in the last 3 days in your default Dev Hub org; show limited details:

```
sf package version list --concise --created-last-days 3 --released
```

List released package versions that were modified today for the two packages with specified aliases in your default Dev Hub org:

```
sf package version list --packages exp-mgr,exp-mgr-util --released --modified-last-days 0
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`

Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-c | --created-last-days CREATED-LAST-DAYS
Optional

Number of days since the request was created, starting at 00:00:00 of first day to now. Use 0 for today.

Type: option

--concise
Optional

Display limited package version details.

Type: boolean

--show-conversions-only
Optional

Filter the list output to display only converted package version.

Type: boolean

-m | --modified-last-days MODIFIED-LAST-DAYS
Optional

Number of days since the items were modified, starting at 00:00:00 of first day to now. Use 0 for today.

Type: option

-p | --packages PACKAGES
Optional

Comma-delimited list of packages (aliases or 0Ho IDs) to list.

Type: option

-r | --released
Optional

Display released versions only (IsReleased=true).

Type: boolean

-b | --branch BRANCH
Optional

Branch in your source control system used to filter the results; only package versions based on the specified branch are listed.

Type: option

-o | --order-by ORDER-BY
Optional

Package version fields used to order the list.

Type: option

--verbose
Optional

Display extended package version details.

Type: boolean

Aliases for `package version list`

```
force:package:version:list
```

`package version promote`

Promote a package version to released.

Description for `package version promote`

Supply the ID or alias of the package version you want to promote. Promotes the package version to released status.

Examples for `package version promote`

Promote the package version with the specified ID to released; uses your default Dev Hub org:

```
sf package version promote --package 04t...
```

Promote the package version with the specified alias to released; uses the Dev Hub org with username devhub@example.com:

```
sf package version promote --package awesome_package_alias --target-dev-hub
devhub@example.com
```

Promote the package version with an alias that has spaces to released:

```
sf package version promote --package "Awesome Package Alias"
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-v | --target-dev-hub TARGET-DEV-HUB`
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package version to promote.

Type: option

-n | --no-prompt
Optional

Don't prompt to confirm setting the package version as released.

Type: boolean

Aliases for `package version promote`

```
force:package:version:promote
```

`package version report`

Retrieve details about a package version in the Dev Hub org.

Description for `package version report`

To update package version values, run "sf package version update".

Examples for `package version report`

Retrieve details about the package version with the specified ID from your default Dev Hub org:

```
sf package version report --package 04t...
```

Retrieve details about the package version with the specified alias (that contains spaces) from the Dev Hub org with username devhub@example.com:

```
sf package version report --package "Your Package Alias" --target-dev-hub devhub@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package to retrieve details for.

Type: option

--verbose
Optional

Display extended package version details.

Type: boolean

Aliases for `package version report`

```
force:package:version:report
```

`package version update`

Update a package version.

Description for `package version update`

Specify a new value for each option you want to update.

To display details about a package version, run "sf package version display".

Examples for `package version update`

Update the package version that has the specified alias (that contains spaces) with a new installation key "password123"; uses your default Dev Hub org:

```
sf package version update --package "Your Package Alias" --installation-key password123
```

Update the package version that has the specified ID with a new branch and tag; use the Dev Hub org with username devhub@example.com:

```
sf package version update --package 04t... --branch main --tag 'Release 1.0.7'
--target-dev-hub devhub@example.com
```

Update the package version that has the specified ID with a new description:

```
sf package version update --package 04t... --version-description "New Package Version
Description"
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-v | --target-dev-hub TARGET-DEV-HUB
Required

Username or alias of the Dev Hub org. Not required if the `target-dev-hub` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-p | --package PACKAGE
Required

ID (starts with 04t) or alias of the package to update a version of.

Type: option

-a | --version-name VERSION-NAME
Optional

New package version name.

Type: option

-e | --version-description VERSION-DESCRIPTION
Optional

New package version description.

Type: option

-b | --branch BRANCH
Optional

New package version branch.

Type: option

-t | --tag TAG
Optional

New package version tag.

Type: option

-k | --installation-key INSTALLATION-KEY
Optional

New installation key for key-protected package (default: null)

Type: option

Aliases for `package version update`

```
force:package:version:update
```

`package1` Commands

Commands to develop first-generation managed and unmanaged packages.

package1 version create
Create a first-generation package version in the release org.

package1 version create get
Retrieve the status of a package version creation request.

package1 version display
Display details about a first-generation package version.

package1 version list
List package versions for the specified first-generation package or for the org.

`package1 version create`

Create a first-generation package version in the release org.

Description for `package1 version create`

The package version is based on the contents of the specified metadata package. Omit --managed-released if you want to create an unmanaged package version.

Examples for `package1 version create`

Create a first-generation package version from the package with the specified ID and name the package version "example"; use your default org:

```
sf package1 version create --package-id 033... --name example
```

Same as previous example, but provide a description and wait for 30 minutes for the package version to be created; use the specified org:

```
sf package1 version create --package-id 033... --name example --description "example
description" --wait 30 --target-org myorg@example.com
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --package-id PACKAGE-ID
Required

ID of the metadata package (starts with 033) of which you're creating a new version.

Type: option

-n | --name NAME
Required

Package version name.

Type: option

-d | --description DESCRIPTION
Optional

Package version description.

Type: option

-v | --version VERSION
Optional

Package version in major.minor format, for example, 3.2.

Type: option

-m | --managed-released
Optional

Create a managed package version.

To create a beta version, don't include this parameter.

Type: boolean

-r | --release-notes-url RELEASE-NOTES-URL
Optional

Release notes URL.

This link is displayed in the package installation UI to provide release notes for this package version to subscribers.

Type: option

-p | --post-install-url POST-INSTALL-URL
Optional

Post install URL.

The contents of the post-installation instructions URL are displayed in the UI after installation of the package version.

Type: option

-k | --installation-key INSTALLATION-KEY
Optional

Installation key for key-protected package (default: null).

Type: option

-w | --wait WAIT
Optional

Minutes to wait for the package version to be created (default: 2 minutes).

Type: option

Aliases for `package1 version create`

```
force:package1:version:create
```

`package1 version create get`

Retrieve the status of a package version creation request.

Examples for `package1 version create get`

Get the status of the creation request for the package version with the specified ID in your default org:

```
sf package1 version create get --request-id 0HD...
```

Same as previous example, but use the specified org:

```
sf package1 version create get --request-id 0HD... --target-org myorg@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

-i | --request-id REQUEST-ID

Required

ID of the PackageUploadRequest (starts with 0HD).

Type: option

Aliases for `package1 version create get`

```
force:package1:version:create:get
```

package1 version display

Display details about a first-generation package version.

Examples for `package1 version display`

Display details about the first-generation package version with the specified ID in your default org:

```
sf package1 version display --package-version-id 04t...
```

Same as previous example, but use the specified org:

```
sf package1 version display --package-version-id 04t... --target-org myorg@example.com
```

Flags

--json

Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR

Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --package-version-id PACKAGE-VERSION-ID
Required

ID (starts with 04t) of the metadata package version whose details you want to display.

Type: option

Aliases for `package1 version display`

```
force:package1:version:display
```

`package1 version list`

List package versions for the specified first-generation package or for the org.

Examples for `package1 version list`

List all first-generation package versions in your default org:

```
sf package1 version list
```

List package versions for the specified first-generation package in the specifief org:

```
sf package1 version list --package-id 033... --target-org myorg@example.com
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-i | --package-id PACKAGE-ID
Optional

Metadata package ID (starts with 033) whose package versions you want to list.

If not specified, shows all versions for all packages (managed and unmanaged) in the org.

Type: option

Aliases for `package1 version list`

```
force:package1:version:list
```

`plugins` Commands

Find and manage plugins

plugins discover
See a list of 3rd-party sf plugins you can install.

`plugins discover`

See a list of 3rd-party sf plugins you can install.

Examples for `plugins discover`

```
sf plugins discover
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

`project` Commands

Work with projects, such as deploy and retrieve metadata.

project convert mdapi
Convert metadata retrieved via Metadata API into the source format used in Salesforce DX projects.

project convert source
Convert source-formatted files into metadata that you can deploy using Metadata API.

project convert source-behavior (Beta)
Enable a behavior of your project source files, and then update your Salesforce DX project to implement the behavior.

project delete source
Delete source from your project and from a non-source-tracked org.

project delete tracking
Delete all local source tracking information.

project deploy cancel
Cancel a deploy operation.

project deploy pipeline quick (Beta)
Quickly deploy a validated deployment to an org.

project deploy pipeline report (Beta)
Check the status of a pipeline deploy operation.

project deploy pipeline resume (Beta)
Resume watching a pipeline deploy operation.

project deploy pipeline start (Beta)
Deploy changes from a branch to the pipeline stage's org.

project deploy pipeline validate (Beta)
Perform a validate-only deployment from a branch to the pipeline stage's org.

project deploy preview
Preview a deployment to see what will deploy to the org, the potential conflicts, and the ignored files.

project deploy quick
Quickly deploy a validated deployment to an org.

project deploy report
Check or poll for the status of a deploy operation.

project deploy resume
Resume watching a deploy operation and update source tracking when the deploy completes.

project deploy start
Deploy metadata to an org from your local project.

project deploy validate
Validate a metadata deployment without actually executing it.

project generate
Generate a Salesforce DX project.

project generate manifest
Create a project manifest that lists the metadata components you want to deploy or retrieve.

project list ignored
Check your local project package directories for forceignored files.

project reset tracking
Reset local and remote source tracking.

project retrieve preview

Preview a retrieval to see what will be retrieved from the org, the potential conflicts, and the ignored files.

project retrieve start

Retrieve metadata from an org to your local project.

`project convert mdapi`

Convert metadata retrieved via Metadata API into the source format used in Salesforce DX projects.

Description for `project convert mdapi`

To use Salesforce CLI to work with components that you retrieved via Metadata API, first convert your files from the metadata format to the source format using this command.

To convert files from the source format back to the metadata format, run "sf project convert source".

To convert multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project convert mdapi`

Convert metadata formatted files in the specified directory into source formatted files; writes converted files to your default package directory:

```
$ sf project convert mdapi --root-dir path/to/metadata
```

Similar to previous example, but writes converted files to the specified output directory:

```
$ sf project convert mdapi --root-dir path/to/metadata --output-dir path/to/outputdir
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`

Optional

Override the api version used for api requests made by this command

Type: option

`-r | --root-dir ROOT-DIR`

Required

Root directory that contains the Metadata API–formatted metadata.

Type: option

-d | --output-dir OUTPUT-DIR
Optional

Directory to store your files in after they're converted to source format; can be an absolute or relative path.

Type: option

-x | --manifest MANIFEST
Optional

File path to manifest (package.xml) of metadata types to convert.

If you specify this flag, don't specify --metadata or --source-dir.

Type: option

-p | --metadata-dir METADATA-DIR
Optional

Root of directory or zip file of metadata formatted files to convert.

The supplied paths can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all metadata types in the directory and its sub-directories).

If you specify this flag, don't specify --manifest or --metadata. If the comma-separated list you're supplying contains spaces, enclose the entire comma-separated list in one set of double quotes.

Type: option

-m | --metadata METADATA
Optional

Metadata component names to convert.

Type: option

Aliases for `project convert mdapi`

```
force:mdapi:convert
```

`project convert source`

Convert source-formatted files into metadata that you can deploy using Metadata API.

Description for `project convert source`

To convert source-formatted files into the metadata format, so that you can deploy them using Metadata API, run this command. Then deploy the metadata using "sf project deploy".

To convert Metadata API–formatted files into the source format, run "sf project convert mdapi".

To specify a package name that includes spaces, enclose the name in single quotes.

To convert multiple components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project convert source`

Convert source-formatted files in the specified directory into metadata-formatted files; writes converted files into a new directory:

```
$ sf project convert source --root-dir path/to/source
```

Similar to previous example, but writes converted files to the specified output directory and associates the files with the specified package:

```
$ sf project convert source --root-dir path/to/source --output-dir path/to/outputdir
--package-name 'My Package'
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`
Optional

API Version to use in the generated project's manifest. By default, will use the version from sfdx-project.json

Override the api version used for api requests made by this command

Type: option

`-r | --root-dir ROOT-DIR`
Optional

Source directory other than the default package to convert.

Type: option

`-d | --output-dir OUTPUT-DIR`
Optional

Output directory to store the Metadata API–formatted files in.

Type: option

Default value: metadataPackage_1746047059705

`-n | --package-name PACKAGE-NAME`
Optional

Name of the package to associate with the metadata-formatted files.

Type: option

`-x | --manifest MANIFEST`
Optional

Path to the manifest (package.xml) file that specifies the metadata types to convert.

If you specify this flag, don't specify --metadata or --source-dir.

Type: option

-p | --source-dir SOURCE-DIR
Optional

Paths to the local source files to convert.

The supplied paths can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all metadata types in the directory and its sub-directories).

If you specify this flag, don't specify --manifest or --metadata.

Type: option

-m | --metadata METADATA
Optional

Metadata component names to convert.

Type: option

Aliases for `project convert source`

```
force:source:convert
```

`project convert source-behavior` (Beta)

Enable a behavior of your project source files, and then update your Salesforce DX project to implement the behavior.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project convert source-behavior`

Specifically, this command updates the "sourceBehaviorOption" option in the "sfdx-project.json" file and then converts the associated local source files in your project as needed.

For example, run this command with the "--behavior decomposePermissionSetBeta" flag to start decomposing permission sets when you deploy or retrieve them. Decomposing means breaking up the monolithic metadata API format XML file that corresponds to a metadata component into smaller XML files and directories based on its subtypes. Permission sets are not decomposed by default; you must opt-in to start decomposing them by using this command. When the command finishes, your "sfdx-project.json" file is updated to always decompose permission sets, and the existing permission set files in your local package directories are converted into the new decomposed format. You run this command only once for a given behavior change.

For more information about the possible values for the --behavior flag, see the "sourceBehaviorOptions" section in the https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_ws_config.htm topic.

Examples for `project convert source-behavior`

Update your Salesforce DX project to decompose custom permission sets:

```
sf project convert source-behavior --behavior decomposePermissionSetBeta
```

Display what the command would do, but don't change any existing files:

```
sf project convert source-behavior --behavior decomposePermissionSetBeta --dry-run
```

Keep the temporary directory that contains the interim metadata API formatted files:

```
sf project convert source-behavior --behavior decomposePermissionSetBeta --dry-run
--preserve-temp-dir
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-b | --behavior BEHAVIOR
Required

Behavior to enable; the values correspond to the possible values of the "sourceBehaviorOption" option in the "sfdx-project.json" file.

Type: option

Permissible values are: decomposeCustomLabelsBeta2, decomposeCustomLabelsBeta, decomposePermissionSetBeta, decomposePermissionSetBeta2, decomposeSharingRulesBeta, decomposeWorkflowBeta, decomposeExternalServiceRegistrationBeta

--dry-run
Optional

Display what the command would do, but don't make any actual changes.

Type: boolean

--preserve-temp-dir
Optional

Don't delete the metadata API format temporary directory that this command creates. Useful for debugging.

Type: boolean

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org.

Type: option

project delete source

Delete source from your project and from a non-source-tracked org.

Description for project delete source

Use this command to delete components from orgs that don't have source tracking. To remove deleted items from orgs that have source tracking enabled, "sf project deploy start".

When you run this command, both the local source file and the metadata component in the org are deleted.

To delete multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project delete source`

Delete all local Apex source files and all Apex classes from the org with alias "my-scratch":

```
sf project delete source --metadata ApexClass --target-org my-scratch
```

Delete a specific Apex class and a Profile that has a space in it from your default org; don't prompt for confirmation:

```
sf project delete source --metadata ApexClass:MyFabulousApexClass --metadata "Profile: My
 Profile" --no-prompt
```

Run the tests that aren't in any managed packages as part of the deletion; if the delete succeeds, and the org has source-tracking enabled, update the source tracking information:

```
sf project delete source --metadata ApexClass --test-level RunLocalTests --track-source
```

Delete the Apex source files in a directory and the corresponding components from your default org:

```
sf project delete source --source-dir force-app/main/default/classes
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`-c | --check-only`
Optional

Validate delete command but don't delete anything from the org or the local project.

IMPORTANT: Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

Validates the deleted metadata and runs all Apex tests, but prevents the deletion from being saved to the org.

If you change a field type from Master-Detail to Lookup or vice versa, that change isn't supported when using the --check-only flag to test a deletion (validation). This kind of change isn't supported for test deletions to avoid the risk of data loss or corruption. If a change that isn't supported for test deletions is included in a deletion package, the test deletion fails and issues an error.

If your deletion package changes a field type from Master-Detail to Lookup or vice versa, you can still validate the changes prior to deploying to Production by performing a full deletion to another test Sandbox. A full deletion includes a validation of the changes as part of the deletion process.

Note: A Metadata API deletion that includes Master-Detail relationships deletes all detail records in the Recycle Bin in the following cases.

1. For a deletion with a new Master-Detail field, soft delete (send to the Recycle Bin) all detail records before proceeding to delete the Master-Detail field, or the deletion fails. During the deletion, detail records are permanently deleted from the Recycle Bin and cannot be recovered.

2. For a deletion that converts a Lookup field relationship to a Master-Detail relationship, detail records must reference a master record or be soft-deleted (sent to the Recycle Bin) for the deletion to succeed. However, a successful deletion permanently deletes any detail records in the Recycle Bin.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to finish.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you.

Type: option

Default value: 33 minutes

--tests TESTS
Optional

Apex tests to run when --test-level is RunSpecifiedTests.

If a test name contains a space, enclose it in double quotes.

For multiple test names, use one of the following formats:

- Repeat the flag for multiple test names: --tests Test1 --tests Test2 --tests "Test With Space"

- Separate the test names with spaces: --tests Test1 Test2 "Test With Space"

Type: option

-l | --test-level TEST-LEVEL
Optional

Deployment Apex testing level.

Valid values are:

- NoTestRun — No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial orgs. This test level is the default for development environments.

- RunSpecifiedTests — Runs only the tests that you specify with the --tests flag. Code coverage requirements differ from the default coverage requirements when using this test level. Executed tests must comprise a minimum of 75% code coverage for each class and trigger in the deployment package. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default for production deployments that include Apex classes or triggers.

- RunAllTestsInOrg — All tests in your org are run, including tests of managed packages.

If you don't specify a test level, the default behavior depends on the contents of your deployment package and target org. For more information, see "Running Tests in a Deployment" in the Metadata API Developer Guide.

Type: option

Permissible values are: NoTestRun, RunSpecifiedTests, RunLocalTests, RunAllTestsInOrg

-r | --no-prompt
Optional

Don't prompt for delete confirmation.

Type: boolean

-m | --metadata METADATA
Optional

Metadata components to delete.

If you specify this flag, don't specify --source-dir.

Type: option

-p | --source-dir SOURCE-DIR
Optional

Source file paths to delete.

The supplied paths can be a single file (in which case the operation is applied to only one file) or a folder (in which case the operation is applied to all metadata types in the directory and its sub-directories).

If you specify this flag, don't specify --metadata.

Type: option

-t | --track-source
Optional

If the delete succeeds, update the source tracking information.

Type: boolean

-f | --force-overwrite
Optional

Ignore conflict warnings and overwrite changes to the org.

Type: boolean

--verbose
Optional

Verbose output of the delete result.

Type: boolean

Aliases for `project delete source`

```
force:source:delete
```

`project delete tracking`

Delete all local source tracking information.

Description for `project delete tracking`

WARNING: This command deletes or overwrites all existing source tracking files. Use with extreme caution.

Deletes all local source tracking information. When you next run 'project deploy preview', Salesforce CLI displays all local and remote files as changed, and any files with the same name are listed as conflicts.

Examples for `project delete tracking`

Delete local source tracking for the org with alias "my-scratch":

```
$ sf project delete tracking --target-org my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`-p | --no-prompt`
Optional

Don't prompt for source tracking override confirmation.

Type: boolean

Aliases for `project delete tracking`

```
force:source:tracking:clear
```

`project deploy cancel`

Cancel a deploy operation.

Description for `project deploy cancel`

Use this command to cancel a deploy operation that hasn't yet completed in the org. Deploy operations include standard deploys, quick deploys, deploy validations, and deploy cancellations.

Run this command by either passing it a job ID or specifying the --use-most-recent flag to use the job ID of the most recent deploy operation.

Examples for `project deploy cancel`

Cancel a deploy operation using a job ID:

```
sf project deploy cancel --job-id 0Af0x000017yLUFCA2
```

Cancel the most recent deploy operation:

```
sf project deploy cancel --use-most-recent
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Optional

Username or alias of the target org.

Type: option

`--async`
Optional

Run the command asynchronously.

The command immediately returns the control of the terminal to you. This way, you can continue to use the CLI. To resume watching the cancellation, run "sf project deploy resume". To check the status of the cancellation, run "sf project deploy report".

Type: boolean

`-i | --job-id JOB-ID`
Optional

Job ID of the deploy operation you want to cancel.

These commands return a job ID if they time out or you specified the --async flag:

- sf project deploy start

- sf project deploy validate

- sf project deploy quick

- sf project deploy cancel

The job ID is valid for 10 days from when you started the deploy operation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent deploy operation.

For performance reasons, this flag uses job IDs for deploy operations that started only in the past 3 days or less. If your most recent deploy operations was more than 3 days ago, this flag won't find a job ID.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you. To resume watching the cancellation, run "sf project deploy resume". To check the status of the cancellation, run "sf project deploy report".

Type: option

Aliases for `project deploy cancel`

```
deploy:metadata:cancel
```

`project deploy pipeline quick` (Beta)

Quickly deploy a validated deployment to an org.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project deploy pipeline quick`

The first time you run any "project deploy pipeline" command, be sure to authorize the org in which DevOps Center is installed. The easiest way to authorize an org is with the "org login web" command.

Before you run this command, create a validated deployment with the "project deploy pipeline validate" command, which returns a job ID. Validated deployments haven't been deployed to the org yet; you deploy them with this command. Either pass the job ID to this command or use the --use-most-recent flag to use the job ID of the most recently validated deployment. For the quick deploy to succeed, the associated validated deployment must also have succeeded.

Executing this quick deploy command takes less time than a standard deploy because it skips running Apex tests. These tests were previously run as part of the validation. Validating first and then running a quick deploy is useful if the deployment to your production org take several hours and you don’t want to risk a failed deploy.

This command doesn't support source-tracking. The source you deploy overwrites the corresponding metadata in your org. This command doesn’t attempt to merge your source with the versions in your org.

Examples for `project deploy pipeline quick`

Run a quick deploy using your default Devops Center org and a job ID:

```
sf project deploy pipeline quick --job-id 0Af0x000017yLUFCA2
```

Asynchronously run a quick deploy of the most recently validated deployment using an org with alias "my-prod-org":

```
sf project deploy pipeline quick --async --use-most-recent --devops-center-username
my-prod-org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--async`
Optional

Run the command asynchronously.

The command immediately returns the job ID and control of the terminal to you. This way, you can continue to use the CLI. To resume the deployment, run "sf project deploy pipeline resume". To check the status of the deployment, run "sf project deploy pipeline report".

Type: boolean

`--concise`
Optional

Show concise output of the command result.

Type: boolean

`--verbose`
Optional

Show verbose output of the command result.

Type: boolean

`-w | --wait WAIT`
Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To check the status of the operation, run "sf project deploy pipeline report".

Type: option

Default value: 33 minutes

`-c | --devops-center-username DEVOPS-CENTER-USERNAME`
Required

Username or alias of the DevOps Center org.

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the validated deployment to quick deploy.

The job ID is valid for 10 days from when you started the validation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recently validated deployment.

For performance reasons, this flag uses only job IDs that were validated in the past 3 days or less. If your most recent deployment validation was more than 3 days ago, this flag won't find the job ID.

Type: boolean

`project deploy pipeline report` (Beta)

Check the status of a pipeline deploy operation.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project deploy pipeline report`

The first time you run any "project deploy pipeline" command, be sure to authorize the org in which DevOps Center is installed. The easiest way to authorize an org is with the "org login web" command.

Run this command by either indicating a job ID or specifying the —use-most-recent flag to use the job ID of the most recent deploy operation.

Examples for `project deploy pipeline report`

Check the status using a job ID:

```
sf project deploy pipeline report --devops-center-username MyStagingSandbox --job-id
0Af0x000017yLUFCA2
```

Check the status of the most recent deploy operation:

```
sf project deploy pipeline report --devops-center-username MyStagingSandbox --use-most-recent
```

Flags

--json
Optional

Format output as json.

Type: boolean

-c | --devops-center-username DEVOPS-CENTER-USERNAME
Required

Username or alias of the DevOps Center org.

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the pipeline deployment to check the status of.

The job ID is valid for 10 days from when you started the deploy operation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent deploy operation.

For performance reasons, this flag uses job IDs for deploy operations that started in the past 3 days or fewer. If your most recent operation was longer than 3 days ago, this flag won't find the job ID.

Type: boolean

`project deploy pipeline resume` (Beta)

Resume watching a pipeline deploy operation.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project deploy pipeline resume`

The first time you run any "project deploy pipeline" command, be sure to authorize the org in which DevOps Center is installed. The easiest way to authorize an org is with the "org login web" command.

Use this command to resume watching a pipeline deploy operation if the original command times out or you specified the --async flag.

Run this command by either indicating a job ID or specifying the --use-most-recent flag to use the job ID of the most recent deploy operation.

Examples for `project deploy pipeline resume`

Resume watching a deploy operation using a job ID:

```
sf project deploy pipeline resume --job-id 0Af0x000017yLUFCA2
```

Resume watching the most recent deploy operation:

```
sf project deploy pipeline resume --use-most-recent
```

Flags

--json
Optional

Format output as json.

Type: boolean

-c | --devops-center-username DEVOPS-CENTER-USERNAME
Required

Username or alias of the DevOps Center org.

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the pipeline deploy operation you want to resume.

These commands return a job ID if they time out or you specified the --async flag:

- sf project deploy pipeline start

- sf project deploy pipeline validate

- sf project deploy pipeline quick

The job ID is valid for 10 days from when you started the deploy operation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent deploy operation.

For performance reasons, this flag uses job IDs for operations that started in the past 3 days or fewer. If your most recent operation was longer than 3 days ago, this flag won't find a job ID.

Type: boolean

--concise
Optional

Show concise output of the command result.

Type: boolean

--verbose
Optional

Show verbose output of the command result.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To check the status of the operation, run "sf project deploy pipeline report".

Type: option

Default value: 33 minutes

`project deploy pipeline start` (Beta)

Deploy changes from a branch to the pipeline stage's org.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project deploy pipeline start`

The first time you run any "project deploy pipeline" command, be sure to authorize the org in which DevOps Center is installed. The easiest way to authorize an org is with the "org login web" command.

Before you run this command, changes in the pipeline stage's branch must be merged in the source control repository.

Examples for `project deploy pipeline start`

Deploy changes in the Staging branch to the Staging environment (sandbox), if the previous stage is the bundling stage:

```
sf project deploy pipeline start --devops-center-project-name "Recruiting App" --branch-name
 staging --devops-center-username MyStagingSandbox --bundle-version-name 1.0
```

Deploy all changes in the main branch to the release environment:

```
sf project deploy pipeline start --devops-center-project-name "Recruiting App" --branch-name
 main --devops-center-username MyReleaseOrg --deploy-all
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`-b | --branch-name BRANCH-NAME`
Required

Name of the branch in the source control repository that corresponds to the pipeline stage that you want to deploy the changes to.

Type: option

`-v | --bundle-version-name BUNDLE-VERSION-NAME`
Optional

Version name of the bundle.

You must indicate the bundle version if deploying to the environment that corresponds to the first stage after the bundling stage.

Type: option

`-a | --deploy-all`
Optional

Deploy all metadata in the branch to the stage's org.

If you don't specify this flag, only changes in the stage's branch are deployed.

Type: boolean

`-p | --devops-center-project-name DEVOPS-CENTER-PROJECT-NAME`
Required

Name of the DevOps Center project.

Type: option

-c | --devops-center-username DEVOPS-CENTER-USERNAME
Required

Username or alias of the DevOps Center org.

Type: option

-t | --tests TESTS
Optional

Apex tests to run when --test-level is RunSpecifiedTests.

Separate multiple test names with commas. Enclose the entire flag value in double quotes if a test name contains spaces.

Type: option

-l | --test-level TEST-LEVEL
Optional

Deployment Apex testing level.

Valid values are:

- NoTestRun — No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial orgs. This test level is the default for development environments.

- RunSpecifiedTests — Runs only the tests that you specify with the --tests flag. Code coverage requirements differ from the default coverage requirements when using this test level. Executed tests must comprise a minimum of 75% code coverage for each class and trigger in the deployment package. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default for production deployments that include Apex classes or triggers.

- RunAllTestsInOrg — All tests in your org are run, including tests of managed packages.

If you don't specify a test level, the default behavior depends on the contents of your deployment package. For more information, see the section "Running Tests in a Deployment" in the "Metadata API Developer Guide". (https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_deploy_running_tests.htm)

Type: option

Permissible values are: NoTestRun, RunSpecifiedTests, RunLocalTests, RunAllTestsInOrg

--async
Optional

Run the command asynchronously.

The command immediately returns the job ID and control of the terminal to you. This way, you can continue to use the CLI. To resume the deployment, run "sf project deploy pipeline resume". To check the status of the deployment, run "sf project deploy pipeline report".

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To check the status of the operation, run "sf project deploy pipeline report".

Type: option

Default value: 33 minutes

--verbose
Optional

Show verbose output of the command result.

Type: boolean

--concise
Optional

Show concise output of the command result.

Type: boolean

`project deploy pipeline validate` (Beta)

Perform a validate-only deployment from a branch to the pipeline stage's org.

Note: This feature is a Beta Service. Customers may opt to try such Beta Service in its sole discretion. Any use of the Beta Service is subject to the applicable Beta Services Terms provided at Agreements and Terms (https://www.salesforce.com/company/legal/agreements/).

Description for `project deploy pipeline validate`

The first time you run any "project deploy pipeline" command, be sure to authorize the org in which DevOps Center is installed. The easiest way to authorize an org is with the "org login web" command.

A validation runs Apex tests to verify whether a deployment will succeed without actually deploying the metadata to your environment, so you can then quickly deploy the changes later without re-running the tests.

Examples for `project deploy pipeline validate`

Perform a validate-only deployment from the Staging branch to the Staging environment (sandbox):

```
sf project deploy pipeline validate --devops-center-project-name "Recruiting App"
--branch-name staging --devops-center-username MyStagingSandbox
```

Perform a validate-only deployment of all changes from the main branch to the release environment:

```
sf project deploy pipeline validate --devops-center-project-name "Recruiting App"
--branch-name main --devops-center-username MyReleaseOrg --deploy-all
```

Flags

--json
Optional

Format output as json.

Type: boolean

-b | --branch-name BRANCH-NAME
Required

Name of the branch in the source control repository that corresponds to the pipeline stage that you want to deploy the changes to.

Type: option

-v | --bundle-version-name BUNDLE-VERSION-NAME
Optional

Version name of the bundle.

You must indicate the bundle version if deploying to the environment that corresponds to the first stage after the bundling stage.

Type: option

-a | --deploy-all
Optional

Deploy all metadata in the branch to the stage's org.

If you don't specify this flag, only changes in the stage's branch are deployed.

Type: boolean

-p | --devops-center-project-name DEVOPS-CENTER-PROJECT-NAME
Required

Name of the DevOps Center project.

Type: option

-c | --devops-center-username DEVOPS-CENTER-USERNAME
Required

Username or alias of the DevOps Center org.

Type: option

-t | --tests TESTS
Optional

Apex tests to run when --test-level is RunSpecifiedTests.

Separate multiple test names with commas. Enclose the entire flag value in double quotes if a test name contains spaces.

Type: option

-l | --test-level TEST-LEVEL
Optional

Deployment Apex testing level.

Valid values are:

- NoTestRun — No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial orgs. This test level is the default for development environments.

- RunSpecifiedTests — Runs only the tests that you specify with the --tests flag. Code coverage requirements differ from the default coverage requirements when using this test level. Executed tests must comprise a minimum of 75% code coverage for each class and trigger in the deployment package. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default for production deployments that include Apex classes or triggers.

- RunAllTestsInOrg — All tests in your org are run, including tests of managed packages.

If you don't specify a test level, the default behavior depends on the contents of your deployment package. For more information, see the section "Running Tests in a Deployment" in the "Metadata API Developer Guide". (https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_deploy_running_tests.htm)

Type: option

Permissible values are: NoTestRun, RunSpecifiedTests, RunLocalTests, RunAllTestsInOrg

--async
Optional

Run the command asynchronously.

The command immediately returns the job ID and control of the terminal to you. This way, you can continue to use the CLI. To resume the deployment, run "sf project deploy pipeline resume". To check the status of the deployment, run "sf project deploy pipeline report".

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To check the status of the operation, run "sf project deploy pipeline report".

Type: option

Default value: 33 minutes

--verbose
Optional

Show verbose output of the command result.

Type: boolean

--concise
Optional

Show concise output of the command result.

Type: boolean

project deploy preview

Preview a deployment to see what will deploy to the org, the potential conflicts, and the ignored files.

Description for project deploy preview

You must run this command from within a project.

The command outputs a table that describes what will happen if you run the "sf project deploy start" command. The table lists the metadata components that will be deployed and deleted. The table also lists the current conflicts between files in your local project and components in the org. Finally, the table lists the files that won't be deployed because they're included in your .forceignore file.

If your org allows source tracking, then this command displays potential conflicts between the org and your local project. Some orgs, such as production org, never allow source tracking. Source tracking is enabled by default on scratch and sandbox orgs; you can disable source tracking when you create the orgs by specifying the --no-track-source flag on the "sf org create scratch|sandbox" commands.

To preview the deployment of multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project deploy preview`

NOTE: The commands to preview a deployment and actually deploy it use similar flags. We provide a few preview examples here, but see the help for "sf project deploy start" for more examples that you can adapt for previewing.

Preview the deployment of source files in a directory, such as force-app, to your default org:

```
sf project deploy preview  --source-dir force-app
```

Preview the deployment of all Apex classes to an org with alias "my-scratch":

```
sf project deploy preview --metadata ApexClass --target-org my-scratch
```

Preview deployment of a specific Apex class:

```
sf project deploy preview --metadata ApexClass:MyApexClass
```

Preview deployment of all components listed in a manifest:

```
sf project deploy preview --manifest path/to/package.xml
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-c | --ignore-conflicts`
Optional

Don't display conflicts in preview of the deployment.

This flag applies only to orgs that allow source tracking. It has no effect on orgs that don't allow it, such as production orgs.

Type: boolean

`-x | --manifest MANIFEST`
Optional

Full file path for manifest (package.xml) of components to preview.

All child components are included. If you specify this flag, don’t specify --metadata or --source-dir.

Type: option

`-m | --metadata METADATA`
Optional

Metadata component names to preview.

Type: option

`-d | --source-dir SOURCE-DIR`
Optional

Path to the local source files to preview.

The supplied path can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all metadata types in the directory and its subdirectories).

If you specify this flag, don't specify --metadata or --manifest.

Type: option

-o | --target-org TARGET-ORG

Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--concise

Optional

Show only the changes that will be deployed; omits files that are forceignored.

Type: boolean

Aliases for `project deploy preview`

```
deploy:metadata:preview
```

`project deploy quick`

Quickly deploy a validated deployment to an org.

Description for `project deploy quick`

Before you run this command, first create a validated deployment with the "sf project deploy validate" command, which returns a job ID. Validated deployments haven't been deployed to the org yet; you deploy them with this command. Either pass the job ID to this command or use the --use-most-recent flag to use the job ID of the most recently validated deployment. For the quick deploy to succeed, the associated validated deployment must also have succeeded.

Executing this quick deploy command takes less time than a standard deploy because it skips running Apex tests. These tests were previously run as part of the validation. Validating first and then running a quick deploy is useful if the deployment to your production org take several hours and you don't want to risk a failed deploy.

This command doesn't support source-tracking. The source you deploy overwrites the corresponding metadata in your org. This command doesn't attempt to merge your source with the versions in your org.

Note: Don't use this command on sandboxes; the command is intended to be used on production orgs. By default, sandboxes don't run tests during a deploy. Use "sf project deploy start" instead.

Examples for `project deploy quick`

Run a quick deploy to your default org using a job ID:

```
sf project deploy quick --job-id 0Af0x000017yLUFCA2
```

Asynchronously run a quick deploy of the most recently validated deployment to an org with alias "my-prod-org":

```
sf project deploy quick --async --use-most-recent --target-org my-prod-org
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--async
Optional

Run the command asynchronously.

The command immediately returns the control of the terminal to you. This way, you can continue to use the CLI. To resume watching the deploy, run "sf project deploy resume". To check the status of the deploy, run "sf project deploy report".

Type: boolean

--concise
Optional

Show concise output of the deploy result.

Type: boolean

-i | --job-id JOB-ID
Optional

Job ID of the deployment you want to quick deploy.

The job ID is valid for 10 days from when you started the validation.

Type: option

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recently validated deployment.

For performance reasons, this flag uses only job IDs that were validated in the past 3 days or less. If your most recent deployment validation was more than 3 days ago, this flag won't find a job ID.

Type: boolean

--verbose
Optional

Show verbose output of the deploy result.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you. To resume watching the deploy, run "sf project deploy resume". To check the status of the deploy, run "sf project deploy report".

Type: option

Default value: 33 minutes

-a | --api-version API-VERSION
Optional

Target API version for the deploy.

Use this flag to override the default API version with the API version of your package.xml file. The default API version is the latest version supported by the CLI.

Type: option

Aliases for `project deploy quick`

```
deploy:metadata:quick
```

`project deploy report`

Check or poll for the status of a deploy operation.

Description for `project deploy report`

Deploy operations include standard deploys, quick deploys, deploy validations, and deploy cancellations.

Run this command by either passing it a job ID or specifying the --use-most-recent flag to use the job ID of the most recent deploy operation. If you specify the --wait flag, the command polls for the status every second until the timeout of --wait minutes. If you don't specify the --wait flag, the command simply checks and displays the status of the deploy; the command doesn't poll for the status.

You typically don't specify the --target-org flag because the cached job already references the org to which you deployed. But if you run this command on a computer different than the one from which you deployed, then you must specify the --target-org and it must point to the same org.

This command doesn't update source tracking information.

Examples for `project deploy report`

Check the status using a job ID:

```
sf project deploy report --job-id 0Af0x000017yLUFCA2
```

Check the status of the most recent deploy operation:

```
sf project deploy report --use-most-recent
```

Poll for the status using a job ID and target org:

```
sf project deploy report --job-id 0Af0x000017yLUFCA2 --target-org me@my.org --wait 30
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Optional

Username or alias of the target org.

Type: option

-i | --job-id JOB-ID
Optional

Job ID of the deploy operation you want to check the status of.

These commands return a job ID if they time out or you specified the --async flag:

- sf project deploy start

- sf project deploy validate

- sf project deploy quick

- sf project deploy cancel

The job ID is valid for 10 days from when you started the deploy operation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent deploy operation.

For performance reasons, this flag uses job IDs for deploy operations that started only in the past 3 days or less. If your most recent operation was more than 3 days ago, this flag won't find a job ID.

Type: boolean

--coverage-formatters COVERAGE-FORMATTERS
Optional

Format of the code coverage results.

For multiple formatters, repeat the flag for each formatter.

--coverage-formatters lcov --coverage-formatters clover

Type: option

Permissible values are: clover, cobertura, html-spa, html, json, json-summary, lcovonly, none, teamcity, text, text-summary

--junit
Optional

Output JUnit test results.

Type: boolean

--results-dir RESULTS-DIR
Optional

Output directory for code coverage and JUnit results; defaults to the deploy ID.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To resume the deployment, run "sf project deploy resume". To check the status of the deployment, run "sf project deploy report".

Type: option

Aliases for `project deploy report`

```
deploy:metadata:report
```

`project deploy resume`

Resume watching a deploy operation and update source tracking when the deploy completes.

Description for `project deploy resume`

Use this command to resume watching a deploy operation if the original command times out or you specified the --async flag. Deploy operations include standard deploys, quick deploys, deploy validations, and deploy cancellations. This command doesn't resume the original operation itself, because the operation always continues after you've started it, regardless of whether you're watching it or not. When the deploy completes, source tracking information is updated as needed.

Run this command by either passing it a job ID or specifying the --use-most-recent flag to use the job ID of the most recent deploy operation.

Examples for `project deploy resume`

Resume watching a deploy operation using a job ID:

```
sf project deploy resume --job-id 0Af0x000017yLUFCA2
```

Resume watching the most recent deploy operation:

```
sf project deploy resume --use-most-recent
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--concise
Optional

Show concise output of the deploy operation result.

Type: boolean

-i | --job-id JOB-ID
Optional

Job ID of the deploy operation you want to resume.

These commands return a job ID if they time out or you specified the --async flag:

- sf project deploy start

- sf project deploy validate

- sf project deploy quick

- sf project deploy cancel

The job ID is valid for 10 days from when you started the deploy operation.

Type: option

-r | --use-most-recent
Optional

Use the job ID of the most recent deploy operation.

For performance reasons, this flag uses job IDs for deploy operations that started only in the past 3 days or less. If your most recent operation was more than 3 days ago, this flag won't find a job ID.

Type: boolean

--verbose
Optional

Show verbose output of the deploy operation result.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you. To resume watching the deploy operation, run this command again. To check the status of the deploy operation, run "sf project deploy report".

Type: option

--coverage-formatters COVERAGE-FORMATTERS
Optional

Format of the code coverage results.

For multiple formatters, repeat the flag for each formatter.

--coverage-formatters lcov --coverage-formatters clover

Type: option

Permissible values are: clover, cobertura, html-spa, html, json, json-summary, lcovonly, none, teamcity, text, text-summary

--junit

Optional

Output JUnit test results.

Type: boolean

--results-dir RESULTS-DIR

Optional

Output directory for code coverage and JUnit results; defaults to the deploy ID.

Type: option

Aliases for `project deploy resume`

```
deploy:metadata:resume
```

`project deploy start`

Deploy metadata to an org from your local project.

Description for `project deploy start`

You must run this command from within a project.

Metadata components are deployed in source format by default. Deploy them in metadata format by specifying the --metadata-dir flag, which specifies the root directory or ZIP file that contains the metadata formatted files you want to deploy.

If your org allows source tracking, then this command tracks the changes in your source. Some orgs, such as production orgs, never allow source tracking. Source tracking is enabled by default on scratch and sandbox orgs; you can disable source tracking when you create the orgs by specifying the --no-track-source flag on the "sf org create scratch|sandbox" commands.

To deploy multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project deploy start`

Deploy local changes not in the org; uses your default org:

```
sf project deploy start
```

Deploy all source files in the "force-app" directory to an org with alias "my-scratch"; show only concise output, in other words don't print a list of all the source that was deployed:

```
sf project deploy start  --source-dir force-app --target-org my-scratch --concise
```

Deploy all the Apex classes and custom objects that are in the "force-app" directory. The list views, layouts, etc, that are associated with the custom objects are also deployed. Both examples are equivalent:

```
sf project deploy start --source-dir force-app/main/default/classes
force-app/main/default/objects
```

```
sf project deploy start --source-dir force-app/main/default/classes --source-dir
force-app/main/default/objects
```

Deploy all Apex classes that are in all package directories defined in the "sfdx-project.json" file:

```
sf project deploy start --metadata ApexClass
```

Deploy a specific Apex class; ignore any conflicts between the local project and org (be careful with this flag, because it will overwrite the Apex class in the org if there are conflicts!):

```
sf project deploy start --metadata ApexClass:MyApexClass --ignore-conflicts
```

Deploy specific Apex classes that match a pattern; in this example, deploy Apex classes whose names contain the string "MyApex". Also ignore any deployment warnings (again, be careful with this flag! You typically want to see the warnings):

```
sf project deploy start --metadata 'ApexClass:MyApex*' --ignore-warnings
```

Deploy a custom object called ExcitingObject that's in the SBQQ namespace:

```
sf project deploy start --metadata CustomObject:SBQQ__ExcitingObject
```

Deploy all custom objects in the SBQQ namespace by using a wildcard and quotes:

```
sf project deploy start --metadata 'CustomObject:SBQQ__*'
```

Deploy all custom objects and Apex classes found in all defined package directories (both examples are equivalent):

```
sf project deploy start --metadata CustomObject ApexClass
```

```
sf project deploy start --metadata CustomObject --metadata ApexClass
```

Deploy all Apex classes and a profile that has a space in its name:

```
sf project deploy start --metadata ApexClass --metadata "Profile:My Profile"
```

Deploy all components listed in a manifest:

```
sf project deploy start --manifest path/to/package.xml
```

Run the tests that aren't in any managed packages as part of a deployment:

```
sf project deploy start --metadata ApexClass --test-level RunLocalTests
```

Deploy all metadata formatted files in the "MDAPI" directory:

```
sf project deploy start --metadata-dir MDAPI
```

Deploy all metadata formatted files in the "MDAPI" directory; items listed in the MDAPI/destructiveChangesPre.xml and MDAPI/destructiveChangesPost.xml manifests are immediately eligible for deletion rather than stored in the Recycle Bin:

```
sf project deploy start --metadata-dir MDAPI --purge-on-delete
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-a | --api-version API-VERSION
Optional

Target API version for the deploy.

Use this flag to override the default API version with the API version of your package.xml file. The default API version is the latest version supported by the CLI.

Type: option

--async
Optional

Run the command asynchronously.

The command immediately returns the job ID and control of the terminal to you. This way, you can continue to use the CLI. To resume the deployment, run "sf project deploy resume". To check the status of the deployment, run "sf project deploy report".

Type: boolean

--concise
Optional

Show concise output of the deploy result.

Type: boolean

--dry-run
Optional

Validate deploy and run Apex tests but don’t save to the org.

Type: boolean

-c | --ignore-conflicts
Optional

Ignore conflicts and deploy local files, even if they overwrite changes in the org.

This flag applies only to orgs that allow source tracking. It has no effect on orgs that don't allow it, such as production orgs.

Type: boolean

-r | --ignore-errors
Optional

Ignore any errors and don’t roll back deployment.

Never use this flag when deploying to a production org. If you specify it, components without errors are deployed and components with errors are skipped, and could result in an inconsistent production org.

Type: boolean

-g | --ignore-warnings
Optional

Ignore warnings and allow a deployment to complete successfully.

If you specify this flag, and a warning occurs, the success status of the deployment is set to true. If you don't specify this flag, and a warning occurs, then the success status is set to false, and the warning is treated like an error.

This flag is useful in a CI environment and your deployment includes destructive changes; if you try to delete a component that doesn't exist in the org, you get a warning. In this case, to ensure that the command returns a success value of true, specify this flag.

Type: boolean

-x | --manifest MANIFEST
Optional

Full file path for manifest (package.xml) of components to deploy.

All child components are included. If you specify this flag, don’t specify --metadata or --source-dir.

Type: option

-m | --metadata METADATA
Optional

Metadata component names to deploy. Wildcards (`*`) supported as long as you use quotes, such as `ApexClass:MyClass*`.

Type: option

--metadata-dir METADATA-DIR
Optional

Root of directory or zip file of metadata formatted files to deploy.

Type: option

--single-package
Optional

Indicates that the metadata zip file points to a directory structure for a single package.

Type: boolean

-d | --source-dir SOURCE-DIR
Optional

Path to the local source files to deploy.

The supplied path can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all metadata types in the directory and its subdirectories).

If you specify this flag, don’t specify --metadata or --manifest.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-t | --tests TESTS
Optional

Apex tests to run when --test-level is RunSpecifiedTests.

If a test name contains a space, enclose it in double quotes.

For multiple test names, use one of the following formats:

- Repeat the flag for multiple test names: --tests Test1 --tests Test2 --tests "Test With Space"

- Separate the test names with spaces: --tests Test1 Test2 "Test With Space"

Type: option

-l | --test-level TEST-LEVEL

Optional

Deployment Apex testing level.

Valid values are:

- NoTestRun — No tests are run. This test level applies only to deployments to development environments, such as sandbox, Developer Edition, or trial orgs. This test level is the default for development environments.

- RunSpecifiedTests — Runs only the tests that you specify with the --tests flag. Code coverage requirements differ from the default coverage requirements when using this test level. Executed tests must comprise a minimum of 75% code coverage for each class and trigger in the deployment package. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default for production deployments that include Apex classes or triggers.

- RunAllTestsInOrg — All tests in your org are run, including tests of managed packages.

If you don't specify a test level, the default behavior depends on the contents of your deployment package and target org. For more information, see [Running Tests in a Deployment](https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_deploy_running_tests.htm) in the "Metadata API Developer Guide".

Type: option

Permissible values are: NoTestRun, RunSpecifiedTests, RunLocalTests, RunAllTestsInOrg

--verbose

Optional

Show verbose output of the deploy result.

Type: boolean

-w | --wait WAIT

Optional

Number of minutes to wait for command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To resume the deployment, run "sf project deploy resume". To check the status of the deployment, run "sf project deploy report".

Type: option

Default value: 33 minutes

--purge-on-delete

Optional

Specify that deleted components in the destructive changes manifest file are immediately eligible for deletion rather than being stored in the Recycle Bin.

Type: boolean

--pre-destructive-changes PRE-DESTRUCTIVE-CHANGES
Optional

File path for a manifest (destructiveChangesPre.xml) of components to delete before the deploy.

Type: option

--post-destructive-changes POST-DESTRUCTIVE-CHANGES
Optional

File path for a manifest (destructiveChangesPost.xml) of components to delete after the deploy.

Type: option

--coverage-formatters COVERAGE-FORMATTERS
Optional

Format of the code coverage results.

For multiple formatters, repeat the flag for each formatter.

--coverage-formatters lcov --coverage-formatters clover

Type: option

Permissible values are: clover, cobertura, html-spa, html, json, json-summary, lcovonly, none, teamcity, text, text-summary

--junit
Optional

Output JUnit test results.

Type: boolean

--results-dir RESULTS-DIR
Optional

Output directory for code coverage and JUnit results; defaults to the deploy ID.

Type: option

Aliases for `project deploy start`

```
deploy:metadata
```

`project deploy validate`

Validate a metadata deployment without actually executing it.

Description for `project deploy validate`

Use this command to verify whether a deployment will succeed without actually deploying the metadata to your org. This command is similar to "sf project deploy start", except you're required to run Apex tests, and the command returns a job ID rather than executing the deployment. If the validation succeeds, then you pass this job ID to the "sf project deploy quick" command to actually deploy the metadata. This quick deploy takes less time because it skips running Apex tests. The job ID is valid for 10 days from when you started the validation. Validating first is useful if the deployment to your production org take several hours and you don't want to risk a failed deploy.

You must run this command from within a project.

This command doesn't support source-tracking. When you quick deploy with the resulting job ID, the source you deploy overwrites the corresponding metadata in your org.

To validate the deployment of multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Note: Don't use this command on sandboxes; the command is intended to be used on production orgs. By default, sandboxes don't run tests during a deploy. If you want to validate a deployment with tests on a sandbox, use "sf project deploy start --dry-run --test-level RunLocalTests" instead.

Examples for `project deploy validate`

NOTE: These examples focus on validating large deployments. See the help for "sf project deploy start" for examples of deploying smaller sets of metadata which you can also use to validate.

Validate the deployment of all source files in the "force-app" directory to the default org:

```
sf project deploy validate --source-dir force-app
```

Validate the deployment of all source files in two directories: "force-app" and "force-app-utils":

```
sf project deploy validate --source-dir force-app --source-dir force-app-utils
```

Asynchronously validate the deployment and run all tests in the org with alias "my-prod-org"; command immediately returns the job ID:

```
sf project deploy validate --source-dir force-app --async --test-level RunAllTestsInOrg
--target-org my-prod-org
```

Validate the deployment of all components listed in a manifest:

```
sf project deploy validate --manifest path/to/package.xml
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-a | --api-version API-VERSION`
Optional

Target API version for the validation.

Use this flag to override the default API version with the API version of your package.xml file. The default API version is the latest version supported by the CLI.

Type: option

--async
Optional

Run the command asynchronously.

The command immediately returns the job ID and control of the terminal to you. This way, you can continue to use the CLI. To resume watching the validation, run "sf project deploy resume". To check the status of the validation, run "sf project deploy report".

Type: boolean

--concise
Optional

Show concise output of the validation result.

Type: boolean

-x | --manifest MANIFEST
Optional

Full file path for manifest (package.xml) of components to validate for deployment.

All child components are included. If you specify this flag, don't specify --metadata or --source-dir.

Type: option

-m | --metadata METADATA
Optional

Metadata component names to validate for deployment.

Type: option

-d | --source-dir SOURCE-DIR
Optional

Path to the local source files to validate for deployment.

The supplied path can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all metadata types in the directory and its subdirectories).

If you specify this flag, don't specify --metadata or --manifest.

Type: option

--metadata-dir METADATA-DIR
Optional

Root of directory or zip file of metadata formatted files to deploy.

Type: option

--single-package
Optional

Indicates that the metadata zip file points to a directory structure for a single package.

Type: boolean

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-t | --tests TESTS
Optional

Apex tests to run when --test-level is RunSpecifiedTests.

If a test name contains a space, enclose it in double quotes.

For multiple test names, use one of the following formats:

- Repeat the flag for multiple test names: --tests Test1 --tests Test2 --tests "Test With Space"

- Separate the test names with spaces: --tests Test1 Test2 "Test With Space"

Type: option

-l | --test-level TEST-LEVEL
Optional

Deployment Apex testing level.

Valid values are:

- RunSpecifiedTests — Runs only the tests that you specify with the --tests flag. Code coverage requirements differ from the default coverage requirements when using this test level. Executed tests must comprise a minimum of 75% code coverage for each class and trigger in the deployment package. This coverage is computed for each class and trigger individually and is different than the overall coverage percentage.

- RunLocalTests — All tests in your org are run, except the ones that originate from installed managed and unlocked packages. This test level is the default.

- RunAllTestsInOrg — All tests in your org are run, including tests of managed packages.

Type: option

Permissible values are: RunAllTestsInOrg, RunLocalTests, RunSpecifiedTests

Default value: RunLocalTests

--verbose
Optional

Show verbose output of the validation result.

Type: boolean

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you and returns the job ID. To resume watching the validation, run "sf project deploy resume". To check the status of the validation, run "sf project deploy report".

Type: option

Default value: 33 minutes

-g | --ignore-warnings
Optional

Ignore warnings and allow a deployment to complete successfully.

If you specify this flag, and a warning occurs, the success status of the deployment is set to true. If you don't specify this flag, and a warning occurs, then the success status is set to false, and the warning is treated like an error.

This flag is useful in a CI environment and your deployment includes destructive changes; if you try to delete a component that doesn't exist in the org, you get a warning. In this case, to ensure that the command returns a success value of true, specify this flag.

Type: boolean

--coverage-formatters COVERAGE-FORMATTERS
Optional

Format of the code coverage results.

For multiple formatters, repeat the flag for each formatter.

--coverage-formatters lcov --coverage-formatters clover

Type: option

Permissible values are: clover, cobertura, html-spa, html, json, json-summary, lcovonly, none, teamcity, text, text-summary

--junit
Optional

Output JUnit test results.

Type: boolean

--results-dir RESULTS-DIR
Optional

Output directory for code coverage and JUnit results; defaults to the deploy ID.

Type: option

--purge-on-delete
Optional

Specify that deleted components in the destructive changes manifest file are immediately eligible for deletion rather than being stored in the Recycle Bin.

Type: boolean

--pre-destructive-changes PRE-DESTRUCTIVE-CHANGES
Optional

File path for a manifest (destructiveChangesPre.xml) of components to delete before the deploy

Type: option

--post-destructive-changes POST-DESTRUCTIVE-CHANGES
Optional

File path for a manifest (destructiveChangesPost.xml) of components to delete after the deploy.

Type: option

Aliases for `project deploy validate`

```
deploy:metadata:validate
```

`project generate`

Generate a Salesforce DX project.

Description for `project generate`

A Salesforce DX project has a specific structure and a configuration file (sfdx-project.json) that identifies the directory as a Salesforce DX project. This command generates the necessary configuration files and directories to get you started.

By default, the generated sfdx-project.json file sets the sourceApiVersion property to the default API version currently used by Salesforce CLI. To specify a different version, set the apiVersion configuration variable. For example: "sf config set apiVersion=57.0 --global".

Examples for `project generate`

Generate a project called "mywork":

```
sf project generate --name mywork
```

Similar to previous example, but generate the files in a directory called "myapp":

```
sf project generate --name mywork --default-package-dir myapp
```

Similar to prevoius example, but also generate a default package.xml manifest file:

```
sf project generate --name mywork --default-package-dir myapp --manifest
```

Generate a project with the minimum files and directories:

```
sf project generate --name mywork --template empty
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated project.

Generates a project directory with this name; any valid directory name is accepted. Also sets the "name" property in the sfdx-project.json file to this name.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for project creation.

The template determines the sample configuration files and directories that this command generates. For example, the empty template provides these files and directory to get you started.

- .forceignore

- config/project-scratch-def.json

- sfdx-project.json

- package.json

- force-app (basic source directory structure)

The standard template provides a complete force-app directory structure so you know where to put your source. It also provides additional files and scripts, especially useful when using Salesforce Extensions for VS Code. For example:

- .gitignore: Use Git for version control.

- .prettierrc and .prettierignore: Use Prettier to format your Aura components.

- .vscode/extensions.json: When launched, Visual Studio Code, prompts you to install the recommended extensions for your project.

- .vscode/launch.json: Configures Replay Debugger.

- .vscode/settings.json: Additional configuration settings.

The analytics template provides similar files and the force-app/main/default/waveTemplates directory.

Type: option

Permissible values are: standard, empty, analytics

Default value: standard

-d | --output-dir OUTPUT-DIR
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

-s | --namespace NAMESPACE
Optional

Namespace associated with this project and any connected scratch orgs.

Type: option

-p | --default-package-dir DEFAULT-PACKAGE-DIR
Optional

Default package directory name.

Metadata items such as classes and Lightning bundles are placed inside this folder.

Type: option

Default value: force-app

-x | --manifest
Optional

Generate a manifest (package.xml) for change-set based development.

Generates a default manifest (package.xml) for fetching Apex, Visualforce, Lightning components, and static resources.

Type: boolean

--api-version API-VERSION
Optional

Will set this version as sourceApiVersion in the sfdx-project.json file

Override the api version used for api requests made by this command

Type: option

Aliases for `project generate`

```
force:project:create
```

`project generate manifest`

Create a project manifest that lists the metadata components you want to deploy or retrieve.

Description for `project generate manifest`

Create a manifest from a list of metadata components (--metadata) or from one or more local directories that contain source files (--source-dir). You can specify either of these flags, not both.

Use --type to specify the type of manifest you want to create. The resulting manifest files have specific names, such as the standard package.xml or destructiveChanges.xml to delete metadata. Valid values for this flag, and their respective file names, are:

* package : package.xml (default)

* pre : destructiveChangesPre.xml

* post : destructiveChangesPost.xml

* destroy : destructiveChanges.xml

See https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_deploy_deleting_files.htm for information about these destructive manifest files.

Use --name to specify a custom name for the generated manifest if the pre-defined ones don't suit your needs. You can specify either --type or --name, but not both.

To include multiple metadata components, either set multiple --metadata <name> flags or a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --include-packages and --source-dir.

To build a manifest from the metadata in an org, use the --from-org flag. You can combine --from-org with the --metadata flag to include only certain metadata types, or with the --excluded-metadata flag to exclude certain metadata types. When building a manifest from an org, the command makes many concurrent API calls to discover the metadata that exists in the org. To limit the number of concurrent requests, use the SF_LIST_METADATA_BATCH_SIZE environment variable and set it to a size that works best for your org and environment. If you experience timeouts or inconsistent manifest contents, then setting this environment variable can improve accuracy. However, the command takes longer to run because it sends fewer requests at a time.

Examples for `project generate manifest`

Create a manifest for deploying or retrieving all Apex classes and custom objects:

```
$ sf project generate manifest --metadata ApexClass --metadata CustomObject
```

Create a manifest for deleting the specified Apex class:

```
$ sf project generate manifest --metadata ApexClass:MyApexClass --type destroy
```

Create a manifest for deploying or retrieving all the metadata components in the specified local directory; name the file myNewManifest.xml:

```
$ sf project generate manifest --source-dir force-app --name myNewManifest
```

Create a manifest from the metadata components in the specified org and include metadata in any unlocked packages:

```
$ sf project generate manifest --from-org test@myorg.com --include-packages unlocked
```

Create a manifest from specific metadata types in an org:

```
$ sf project generate manifest --from-org test@myorg.com --metadata
ApexClass,CustomObject,CustomLabels
```

Create a manifest from all metadata components in an org excluding specific metadata types:

```
$ sf project generate manifest --from-org test@myorg.com --excluded-metadata StandardValueSet
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-m | --metadata METADATA
Optional

Names of metadata components to include in the manifest.

Type: option

-p | --source-dir SOURCE-DIR
Optional

Paths to the local source files to include in the manifest.

Type: option

-n | --name NAME
Optional

Name of a custom manifest file to create.

Type: option

-t | --type TYPE
Optional

Type of manifest to create; the type determines the name of the created file.

Type: option

Permissible values are: pre, post, destroy, package

-c | --include-packages INCLUDE-PACKAGES
Optional

Package types (managed, unlocked) whose metadata is included in the manifest; by default, metadata in managed and unlocked packages is excluded. Metadata in unmanaged packages is always included.

Type: option

Permissible values are: managed, unlocked

--excluded-metadata EXCLUDED-METADATA

Optional

Metadata types to exclude when building a manifest from an org. Specify the name of the type, not the name of a specific component.

Type: option

--from-org FROM-ORG

Optional

Username or alias of the org that contains the metadata components from which to build a manifest.

Type: option

-d | --output-dir OUTPUT-DIR

Optional

Directory to save the created manifest.

Type: option

Aliases for `project generate manifest`

```
force:source:manifest:create
```

`project list ignored`

Check your local project package directories for forceignored files.

Description for `project list ignored`

When deploying or retrieving metadata between your local project and an org, you can specify the source files you want to exclude with a .forceignore file. The .forceignore file structure mimics the .gitignore structure. Each line in .forceignore specifies a pattern that corresponds to one or more files. The files typically represent metadata components, but can be any files you want to exclude, such as LWC configuration JSON files or tests.

Examples for `project list ignored`

List all the files in all package directories that are ignored:

```
sf project list ignored
```

List all the files in a specific directory that are ignored:

```
sf project list ignored --source-dir force-app
```

Check if a particular file is ignored:

```
sf project list ignored --source-dir package.xml
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-p | --source-dir SOURCE-DIR
Optional

File or directory of files that the command checks for foreceignored files.

Type: option

Aliases for `project list ignored`

```
force:source:ignored:list
```

`project reset tracking`

Reset local and remote source tracking.

Description for `project reset tracking`

WARNING: This command deletes or overwrites all existing source tracking files. Use with extreme caution.

Resets local and remote source tracking so that Salesforce CLI no longer registers differences between your local files and those in the org. When you next run 'project deploy preview', Salesforce CLI returns no results, even though conflicts might actually exist. Salesforce CLI then resumes tracking new source changes as usual.

Use the --revision flag to reset source tracking to a specific revision number of an org source member. To get the revision number, query the SourceMember Tooling API object with the 'data soql' command. For example:

sf data query --query "SELECT MemberName, MemberType, RevisionCounter FROM SourceMember" --use-tooling-api --target-org my-scratch

Examples for `project reset tracking`

Reset source tracking for the org with alias "my-scratch":

```
$ sf project reset tracking --target-org my-scratch
```

Reset source tracking to revision number 30 for your default org:

```
$ sf project reset tracking --revision 30
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

--api-version API-VERSION
Optional

Override the api version used for api requests made by this command

Type: option

-r | --revision REVISION
Optional

SourceMember revision counter number to reset to.

Type: option

-p | --no-prompt
Optional

Don't prompt for source tracking override confirmation.

Type: boolean

Aliases for `project reset tracking`

```
force:source:tracking:reset
```

`project retrieve preview`

Preview a retrieval to see what will be retrieved from the org, the potential conflicts, and the ignored files.

Description for `project retrieve preview`

You must run this command from within a project.

The command outputs a table that describes what will happen if you run the "sf project retrieve start" command. The table lists the metadata components that will be retrieved and deleted. The table also lists the current conflicts between files in your local project and components in the org. Finally, the table lists the files that won't be retrieved because they're included in your .forceignore file.

If your org allows source tracking, then this command displays potential conflicts between the org and your local project. Some orgs, such as production org, never allow source tracking. Source tracking is enabled by default on scratch and sandbox orgs; you can disable source tracking when you create the orgs by specifying the --no-track-source flag on the "sf org create scratch|sandbox" commands.

Examples for `project retrieve preview`

Preview the retrieve of all changes from your default org:

```
sf project retrieve preview
```

Preview the retrieve when ignoring any conflicts from an org with alias "my-scratch":

```
sf project retrieve preview --ignore-conflicts --target-org my-scratch
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-c | --ignore-conflicts`
Optional

Don't display conflicts in the preview of the retrieval.

This flag applies only to orgs that allow source tracking. It has no effect on orgs that don't allow it, such as production orgs.

Type: boolean

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--concise`
Optional

Show only the changes that will be retrieved; omits files that are forceignored.

Type: boolean

Aliases for `project retrieve preview`

```
retrieve:metadata:preview
```

`project retrieve start`

Retrieve metadata from an org to your local project.

Description for `project retrieve start`

You must run this command from within a project.

Metadata components are retrieved in source format by default. Retrieve them in metadata format by specifying the --target-metadata-dir flag, which retrieves the components into a ZIP file in the specified directory.

If your org allows source tracking, then this command tracks the changes in your source. Some orgs, such as production orgs, never allow source tracking. Source tracking is enabled by default on scratch and sandbox orgs; you can disable source tracking when you create the orgs by specifying the --no-track-source flag on the "sf org create scratch|sandbox" commands.

To retrieve multiple metadata components, either use multiple --metadata <name> flags or use a single --metadata flag with multiple names separated by spaces. Enclose names that contain spaces in one set of double quotes. The same syntax applies to --source-dir.

Examples for `project retrieve start`

Retrieve all remote changes from your default org:

```
sf project retrieve start
```

Retrieve the source files in the "force-app" directory from an org with alias "my-scratch":

```
sf project retrieve start --source-dir force-app --target-org my-scratch
```

Retrieve all the Apex classes and custom objects whose source is in the "force-app" directory. The list views, layouts, etc, that are associated with the custom objects are also retrieved. Both examples are equivalent:

```
sf project retrieve start --source-dir force-app/main/default/classes
force-app/main/default/objects
```

```
sf project retrieve start --source-dir force-app/main/default/classes --source-dir
force-app/main/default/objects
```

Retrieve all Apex classes that are in all package directories defined in the "sfdx-project.json" file:

```
sf project retrieve start --metadata ApexClass
```

Retrieve a specific Apex class; ignore any conflicts between the local project and org (be careful with this flag, because it will overwrite the Apex class source files in your local project if there are conflicts!):

```
sf project retrieve start --metadata ApexClass:MyApexClass --ignore-conflicts
```

Retrieve specific Apex classes that match a pattern; in this example, retrieve Apex classes whose names contain the string "MyApex":

```
sf project retrieve start --metadata 'ApexClass:MyApex*'
```

Retrieve a custom object called ExcitingObject that's in the SBQQ namespace:

```
sf project retrieve start --metadata CustomObject:SBQQ__ExcitingObject
```

Retrieve all custom objects in the SBQQ namespace by using a wildcard and quotes:

```
sf project retrieve start --metadata 'CustomObject:SBQQ__*'
```

Retrieve all list views for the Case standard object:

```
sf project retrieve start --metadata 'ListView:Case*'
```

Retrieve all custom objects and Apex classes found in all defined package directories (both examples are equivalent):

```
sf project retrieve start --metadata CustomObject ApexClass
```

```
sf project retrieve start --metadata CustomObject --metadata ApexClass
```

Retrieve all metadata components listed in a manifest:

```
sf project retrieve start --manifest path/to/package.xml
```

Retrieve metadata from a package:

```
sf project retrieve start --package-name MyPackageName
```

Retrieve metadata from multiple packages, one of which has a space in its name (both examples are equivalent):

```
sf project retrieve start --package-name Package1 "PackageName With Spaces" Package3
```

```
sf project retrieve start --package-name Package1 --package-name "PackageName With Spaces"
 --package-name Package3
```

Retrieve the metadata components listed in the force-app directory, but retrieve them in metadata format into a ZIP file in the "output" directory:

```
sf project retrieve start --source-dir force-app --target-metadata-dir output
```

Retrieve in metadata format and automatically extract the contents into the "output" directory:

```
sf project retrieve start --source-dir force-app --target-metadata-dir output --unzip
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-a | --api-version API-VERSION
Optional

Target API version for the retrieve.

Use this flag to override the default API version, which is the latest version supported the CLI, with the API version in your package.xml file.

Type: option

-c | --ignore-conflicts
Optional

Ignore conflicts and retrieve and save files to your local filesystem, even if they overwrite your local changes.

This flag applies only to orgs that allow source tracking. It has no effect on orgs that don't allow it, such as production orgs.

Type: boolean

-x | --manifest MANIFEST
Optional

File path for the manifest (package.xml) that specifies the components to retrieve.

If you specify this flag, don’t specify --metadata or --source-dir.

Type: option

-m | --metadata METADATA
Optional

Metadata component names to retrieve. Wildcards (`*`) supported as long as you use quotes, such as `ApexClass:MyClass*`.

Type: option

-n | --package-name PACKAGE-NAME
Optional

Package names to retrieve. Use of this flag is for reference only; don't use it to retrieve packaged metadata for development.

The metadata of the supplied package name(s) will be retrieved into a child directory of the project. The name of that child directory matches the name of the package. The retrieved metadata is meant for your reference only, don't add it to a source control system for development and deployment. For package development, retrieve the metadata using a manifest (`--manifest` flag) or by targeting a source controlled package directory within your project (`--source-dir` flag).

Type: option

-r | --output-dir OUTPUT-DIR
Optional

Directory root for the retrieved source files.

The root of the directory structure into which the source files are retrieved.

If the target directory matches one of the package directories in your sfdx-project.json file, the command fails.

Running the command multiple times with the same target adds new files and overwrites existing files.

Type: option

--single-package
Optional

Indicates that the zip file points to a directory structure for a single package.

Type: boolean

-d | --source-dir SOURCE-DIR
Optional

File paths for source to retrieve from the org.

The supplied paths can be to a single file (in which case the operation is applied to only one file) or to a folder (in which case the operation is applied to all source files in the directory and its subdirectories).

Type: option

-t | --target-metadata-dir TARGET-METADATA-DIR
Optional

Directory that will contain the retrieved metadata format files or ZIP.

Type: option

-o | --target-org TARGET-ORG
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

-w | --wait WAIT
Optional

Number of minutes to wait for the command to complete and display results to the terminal window.

If the command continues to run after the wait period, the CLI returns control of the terminal window to you.

Type: option

Default value: 33 minutes

-z | --unzip
Optional

Extract all files from the retrieved zip file.

Type: boolean

--zip-file-name ZIP-FILE-NAME
Optional

File name to use for the retrieved zip file.

Type: option

Aliases for `project retrieve start`

```
retrieve:metadata
```

`scanner` Commands

Scan code to detect code quality issues and security vulnerabilities.

scanner rule add
Add custom rules to Salesforce Code Analyzer's registry to run them along with the built-in rules.

scanner rule describe
Provide detailed information about a rule that includes the rule's language (such as Apex or Java), the violation it detects, example code of the violation, and the rule's categories and rulesets.

scanner rule list
List basic information about all rules matching provided criteria.

scanner rule remove
Remove custom rules from the registry of available rules.

scanner run
Scan a codebase with all the rules in the registry, or use parameters to filter the rules based on rulename, category, or ruleset.

scanner run dfa
Scan codebase with all DFA rules by default.

`scanner rule add`

Add custom rules to Salesforce Code Analyzer's registry to run them along with the built-in rules.

Description for `scanner rule add`

Bundle custom PMD rules in JAR files. Follow PMD conventions, such as defining the custom rules in XML files under a `/category/` directory. Compile and test custom rules separately before adding them. See PMD's documentation for more information on writing rules.

Examples for `scanner rule add`

This example shows how to specify two JAR files directly.

```
sf scanner rule add --language apex --path
"/Users/me/rules/Jar1.jar,/Users/me/rules/Jar2.jar"
```

This example shows how to specify a directory containing one or more JARs, all of which are added to the registry.

```
sf scanner rule add --language apex --path "/Users/me/rules"
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`-l | --language LANGUAGE`
Required

Language that the custom rules are evaluated against.

Type: option

`-p | --path PATH`
Required

One or more paths (such as a directory or JAR file) to custom rule definitions.

Specify multiple values as a comma-separated list.

Type: option

`scanner rule describe`

Provide detailed information about a rule that includes the rule's language (such as Apex or Java), the violation it detects, example code of the violation, and the rule's categories and rulesets.

Examples for `scanner rule describe`

This example shows how to describe the ApexBadCrypto rule.

```
sf scanner rule describe --rulename ApexBadCrypto
```

Flags

--json
Optional

Format output as json.

Type: boolean

-n | --rulename RULENAME
Required

The name of the rule.

Type: option

--verbose
Optional

Emit additional command output to stdout.

Type: boolean

scanner rule list

List basic information about all rules matching provided criteria.

Description for scanner rule list

Filter the output to view a smaller set of rules. Use the `scanner rule describe` command to get information about a specific rule.

Examples for scanner rule list

This example invokes the command without filter criteria, which returns all rules.

```
sf scanner rule list
```

This example returns all rules for Apex OR Javascript. Values supplied to a single filter are handled with a logical OR.

```
sf scanner rule list --language apex,javascript
```

This example returns all rules that target Apex OR Javascript, AND are members of the Braces OR Security rulesets. The different filters are combined with a logical AND.

```
sf scanner rule list --language apex,javascript --ruleset Braces,Security
```

Flags

--json
Optional

Format output as json.

Type: boolean

--verbose
Optional

Emit additional command output to stdout.

Type: boolean

-c | --category CATEGORY
Optional

Select rules by category.

Specify multiple values as a comma-separated list.

Type: option

-r | --ruleset RULESET
The parameter `--ruleset` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

Deprecated. Use category instead. Select rules by ruleset.

Type: option

-l | --language LANGUAGE
Optional

Select rules by language.

Specify multiple values as a comma-separated list.

Type: option

-e | --engine ENGINE
Optional

Select rules by engine.

Specify multiple values as a comma-separated list.

Type: option

Permissible values are: eslint, eslint-lwc, eslint-typescript, pmd, pmd-appexchange, retire-js, cpd, sfge

`scanner rule remove`

Remove custom rules from the registry of available rules.

Description for `scanner rule remove`

Use the `-p|--path` parameter to specify one or more paths to remove. If you don't specify any parameters, the command lists all valid custom paths but doesn't remove any.

Examples for `scanner rule remove`

This example runs the command without arguments to see a list of registered custom paths.

```
sf scanner rule remove
```

This example uses the --path parameter to deregister the rules defined in somerules.jar and any JARs/XMLs contained in the rules folder.

```
sf scanner rule remove --path "~/path/to/somerules.jar,~/path/to/folder/containing/rules"
```

This example uses the --force flag to bypass the confirmation prompt, removing all rules defined in somerules.jar. By default, a list of all rules that will be unregistered is displayed, and the action must be confirmed. To bypass that confirmation, use the --force flag.

```
sf scanner rule remove --force --path "~/path/to/somerules.jar"
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--verbose`
Optional

Emit additional command output to stdout.

Type: boolean

`-f | --force`
Optional

Bypass the confirmation prompt and immediately remove the rules.

Type: boolean

`-p | --path PATH`
Optional

One or more paths to remove.

Specify multiple values with a comma-separated list.

Type: option

`scanner run`

Scan a codebase with all the rules in the registry, or use parameters to filter the rules based on rulename, category, or ruleset.

Description for `scanner run`

Specify the format of the output, such as XML or JUnit. Print the output to the console (default) or to a file using the --outfile parameter.

Examples for `scanner run`

This example evaluates all rules against somefile.js. Invoking code analyzer without specifying any rules causes all rules to be run.

```
sf scanner run --format xml --target "somefile.js"
```

This example evaluates all rules in the Design and Best Practices categories. When you specify multiple categories or rulesets, the results are combined with a logical OR.

```
sf scanner run --format xml --target "somefile.js" --
```

These examples evaluate rules against all .js files in the current directory, except for IgnoreMe.js. Unix example:

```
sf scanner run --target './**/*.js,!./**/IgnoreMe.js' ...
```

Windows example:

```
sf scanner run --target ".\**\*.js,!.\**\IgnoreMe.js" ...
```

This example scans the project contained in '/my-project' if the current working directory is another directory. Specify tsconfig.json if the current working directory does not contain the tsconfig.json that corresponds to the TypeScript files being scanned.

```
sf scanner run --target "/my-project/**/*.ts" --tsconfig "/my-project/tsconfig.json"
```

This example evaluates rules against somefile.js, including Jasmine in the environment variables. Uses --env to override the default ESLint environment variables to add frameworks.

```
sf scanner run --target "somefile.js" --env '{"jasmine": true}'
```

This example evaluates rules aginst somefile.js using eslint-lwc and pmd engines. Use --engine to include or exclude engines. Any engine listed will be run, regardless of its current 'disabled' attribute.

```
sf scanner run --target "somefile.js" --engine "eslint-lwc,pmd"
```

This example executes CPD engine against known file extensions in "/some/dir". CPD helps detect blocks of code duplication in selected languages. Use --engine to invoke engines that are not enabled by default.

```
sf scanner run --target "/some/dir" --engine cpd
```

This example executes rules defined in pmd_rule_ref.xml against the files in 'src'. To use PMD with your own rule reference file, use --pmdconfig. Note that rule filters are not applied.

```
sf scanner run --target "src" --pmdconfig "pmd_rule_ref.xml"
```

This example uses a custom config to scan the files in 'src'. To use ESLint with your own .eslintrc.json file, use --eslintconfig. Make sure that the directory you run the command from has all the NPM dependencies installed.

```
sf scanner run --target "src" --eslintconfig "/home/my/setup/.eslintrc.json"
```

This example uses --normalize-severity to output normalized severity and engine-specific severity across all engines. Normalized severity is: 1 (high), 2 (moderate), and 3 (low).

```
sf scanner run --target "/some-project/" --format csv --normalize-severity
```

This example uses --severity-threshold to throw a non-zero exit code when rule violations of normalized severity 2 or greater are found. If any violations with the specified severity (or greater) are found, the exit code equals the severity of the most severe violation.

```
sf scanner run --target "/some-project/" --severity-threshold 2
```

These examples explain that the paths specified for --projectdir must contain all files specified through --target cumulatively.

```
sf scanner run --target "./myproject/main/default/classes/*.cls" --projectdir "./myproject/"
```

```
sf scanner run --target "./**/*.cls" --projectdir "./"
```

```
sf scanner run --target "./dir1/file1.cls,./dir2/file2.cls" --projectdir "./dir1/,./dir2/"
```

This example fails because the set of files included in --target is larger than that contained in --projectdir.

```
sf scanner run --target "./**/*.cls" --projectdir "./myproject/"
```

Flags

--verbose
Optional

Emit additional command output to stdout.

Type: boolean

-c | --category CATEGORY
Optional

One or more categories of rules to run.

Specify multiple values as a comma-separated list.

Type: option

-f | --format FORMAT
Optional

The output format for results written directly to the console.

Type: option

Permissible values are: csv, html, json, junit, sarif, table, xml

-o | --outfile OUTFILE
Optional

File to write output to.

Type: option

-s | --severity-threshold SEVERITY-THRESHOLD
Optional

An error will be thrown when a violation is found with a severity equal to or greater than the specified level.

Values are 1 (high), 2 (moderate), and 3 (low). Exit code is the most severe violation. Using this flag also invokes the --normalize-severity flag.

Type: option

--normalize-severity
Optional

Include normalized severity levels 1 (high), 2 (moderate), and 3 (low) with the results.

For the html option, the normalized severity is displayed instead of the engine severity.

Type: boolean

-p | --projectdir PROJECTDIR
Optional

The relative or absolute root project directories used to set the context for Graph Engine's analysis.

Specify multiple values as a comma-separated list. Each project directory must be a path, not a glob. If --projectdir isn't specified, a default value is calculated. The default value is a directory that contains all the target files.

Type: option

--json
Optional

Format output as json.

Type: boolean

-r | --ruleset RULESET

The parameter `--ruleset` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

[Deprecated] Rulesets to run.

Use category instead. One or more rulesets to run. Specify multiple values as a comma-separated list.

Type: option

-e | --engine ENGINE

Optional

Specify which engines to run.

Submit multiple values as a comma-separated list.

Type: option

Permissible values are: eslint, eslint-lwc, eslint-typescript, pmd, pmd-appexchange, retire-js, sfge, cpd

-t | --target TARGET

Optional

Source code location.

Specify multiple values as a comma-separated list. Can use glob patterns. Default is ".".

Type: option

--tsconfig TSCONFIG

Optional

Location of tsconfig.json file used by the eslint-typescript engine. The --tsconfig flag can't be used with --eslintconfig flag.

Type: option

--eslintconfig ESLINTCONFIG

Optional

Specify the location of eslintrc config to customize eslint engine. The --tsconfig flag can't be used with --eslintconfig flag.

Type: option

--pmdconfig PMDCONFIG

Optional

Location of PMD rule reference XML file to customize rule selection.

Type: option

--env ENV

The parameter `--env` has been deprecated and will be removed in vNaN.0 or later. Use instead.

Optional

[Deprecated] Override ESLint's default environment variables, in JSON-formatted string.

Overrides ESLint's default environmental variables, in JSON-formatted string.

Type: option

--verbose-violations

Optional

Includes Retire-js violation-message details about each vulnerability in the results, including summary, common vulnerabilities and exposures (CVE), and URLs.

Type: boolean

`scanner run dfa`

Scan codebase with all DFA rules by default.

Description for `scanner run dfa`

Specify the format of output and print results directly or as contents of a file that you provide with --outfile flag.

Examples for `scanner run dfa`

These examples show how the paths specified for --projectdir must contain all files specified through --target cumulatively.

```
sf scanner run dfa --target "./**/*.cls" --projectdir "./"
```

```
sf scanner run dfa --target "./dir1/file1.cls,./dir2/file2.cls" --projectdir
"./dir1/,./dir2/"
```

This example fails because the set of files included in --target is larger than that contained in --projectdir.

```
sf scanner run dfa --target "./**/*.cls" --projectdir "./myproject/"
```

Globs must be wrapped in quotes, as in these Windows and Unix examples, which evaluate rules against all .cls files in the current directory and subdirectories except for IgnoreMe.cls. Unix example:

```
sf scanner run dfa --target "./**/*.cls,!./**/IgnoreMe.cls" ...
```

Windows example:

```
sf scanner run dfa --target ".\**\*.cls,!.\**\IgnoreMe.cls" ...
```

You can target individual methods within a file with a suffix hash (#) on the file's path, and with a semi-colon-delimited list of method names. This syntax is incompatible with globs and directories. This example evaluates rules against all methods named Method1 or Method2 in File1.cls, and all methods named Method3 in File2.cls.

```
sf scanner run dfa --target "./File1.cls#Method1;Method2,./File2.cls#Method3" ...
```

Use --normalize-severity to output a normalized severity across all engines, in addition to the engine-specific severity. Normalized severity is 1 (high), 2 (moderate), and 3 (low).

```
sf scanner run dfa --target "./some-project/" --projectdir "./some-project/" --format csv
 --normalize-severity
```

Use --severity-threshold to throw a non-zero exit code when rule violations of a specific normalized severity or greater are found. If there are any rule violations with a severity of 2 or 1, the exit code is equal to the severity of the most severe violation.

```
sf scanner run dfa --target "./some-project/" --projectdir "./some-project/"
--severity-threshold 2
```

Use --rule-thread-count to allow more (or fewer) entrypoints to be evaluated concurrently.

```
sf scanner run dfa --rule-thread-count 6 ...
```

Use --rule-thread-timeout to increase or decrease the maximum runtime for a single entrypoint evaluation. This increases the timeout from the 15-minute default to 150 minutes.

```
sf scanner run dfa --rule-thread-timeout 9000000 ...
```

Use --sfgejvmargs to pass Java Virtual Machine args to override system defaults while executing Salesforce Graph Engine's rules. The example overrides the system's default heap space allocation to 8 GB and decreases chances of encountering OutOfMemory error.

```
sf scanner run dfa --sfgejvmargs "-Xmx8g" ...
```

Use --with-pilot to allow execution of pilot rules. This example allows pilot rules in the "Performance" category to execute.

```
sf scanner run dfa --category 'Performance' --with-pilot ...
```

Flags

--verbose
Optional

Emit additional command output to stdout.

Type: boolean

-c | --category CATEGORY
Optional

One or more categories of rules to run.

Specify multiple values as a comma-separated list.

Type: option

-f | --format FORMAT
Optional

The output format for results written directly to the console.

Type: option

Permissible values are: csv, html, json, junit, sarif, table, xml

-o | --outfile OUTFILE
Optional

File to write output to.

Type: option

-s | --severity-threshold SEVERITY-THRESHOLD
Optional

An error will be thrown when a violation is found with a severity equal to or greater than the specified level.

Values are 1 (high), 2 (moderate), and 3 (low). Exit code is the most severe violation. Using this flag also invokes the --normalize-severity flag.

Type: option

--normalize-severity
Optional

Include normalized severity levels 1 (high), 2 (moderate), and 3 (low) with the results.

For the html option, the normalized severity is displayed instead of the engine severity.

Type: boolean

-p | --projectdir PROJECTDIR
Optional

The relative or absolute root project directories used to set the context for Graph Engine's analysis.

Specify multiple values as a comma-separated list. Each project directory must be a path, not a glob. If --projectdir isn't specified, a default value is calculated. The default value is a directory that contains all the target files.

Type: option

--json
Optional

Format output as json.

Type: boolean

--with-pilot
Optional

Allow pilot rules to execute.

Type: boolean

-t | --target TARGET
Optional

Source code location.

Use glob patterns or specify individual methods with #-syntax. Multiple values are specified as a comma-separated list. Default is ".".

Type: option

--rule-thread-count RULE-THREAD-COUNT
Optional

Number of DFA rule-evaluation threads or how many entry points can be evaluated concurrently. Inherits its value from the SFGE_RULE_THREAD_COUNT environment variable, if set.

Type: option

--rule-thread-timeout RULE-THREAD-TIMEOUT
Optional

Time limit, in milliseconds, for evaluating a single entry point. Inherits its value from the "SFGE_RULE_THREAD_TIMEOUT" environment variable, if set.

Type: option

--rule-disable-warning-violation
Optional

Disable warning violations from Salesforce Graph Engine. Inherits value from SFGE_RULE_DISABLE_WARNING_VIOLATION env-var, if set.

Examples of warning violations include those on StripInaccessible READ access, to get only high-severity violations.

Type: boolean

--sfgejvmargs SFGEJVMARGS
Optional

Java Virtual Machine (JVM) arguments to override system defaults while executing Salesforce Graph Engine.

Separate multiple arguments by a space.

Type: option

--pathexplimit PATHEXPLIMIT

Optional

Path expansion upper boundary to limit the complexity of code that Graph Engine analyzes before failing test. Inherits it value from the "SFGE_PATH_EXPANSION_LIMIT", if set.

Set the value to -1 to remove any upper boundary. Its default value is derived from JVM heap space allocation.

Type: option

`schema` Commands

Generate metadata files.

schema generate field

Generate metadata source files for a new custom field on a specified object.

schema generate platformevent

Generate metadata source files for a new platform event.

schema generate sobject

Generate metadata source files for a new custom object.

schema generate tab

Generate the metadata source files for a new custom tab on a custom object.

`schema generate field`

Generate metadata source files for a new custom field on a specified object.

Description for `schema generate field`

This command is interactive and must be run in a Salesforce DX project directory. You're required to specify the field's label with the "--label" flag. The command uses this label to provide intelligent suggestions for other field properties, such as its API name.

You can generate a custom field on either a standard object, such as Account, or a custom object. In both cases, the source files for the object must already exist in your local project before you run this command. If you create a relationship field, the source files for the parent object must also exist in your local directory. Use the command "sf metadata retrieve -m CustomObject:<object>" to retrieve source files for both standard and custom objects from your org. To create a custom object, run the "sf generate metadata sobject" command or use the Object Manager UI in your Salesforce org.

Examples for `schema generate field`

Create a field with the specified label; the command prompts you for the object:

```
sf schema generate field --label "My Field"
```

Specify the local path to the object's folder:

```
sf schema generate field --label "My Field" --object
force-app/main/default/objects/MyObject__c
```

Flags

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-l | --label LABEL
Required

The field's label.

Type: option

-o | --object OBJECT
Optional

The directory that contains the object's source files.

The object source files in your local project are grouped in a directoy with the same name as the object. Custom object names always end in "__c". An example of the object directory for the Account standard object is "force-app/main/default/objects/Account" An example custom object directory is "force-app/main/default/objects/MyObject__c"

If you don't specify this flag, the command prompts you to choose from your local objects.

Type: option

Aliases for **schema generate field**

```
generate:metadata:field
```

schema generate platformevent

Generate metadata source files for a new platform event.

Description for **schema generate platformevent**

This command is interactive and must be run in a Salesforce DX project directory. You're required to specify the event's label with the "--label" flag. The command uses this label to provide intelligent suggestions for other event properties, such as its API name.

Examples for **schema generate platformevent**

Create a platform event with the specified label:

```
sf schema generate platformevent --label "My Platform Event"
```

Flags

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-l | --label LABEL
Required

The platform event's label.

Type: option

Aliases for `schema generate platformevent`

```
generate:metadata:platformevent
```

`schema generate sobject`

Generate metadata source files for a new custom object.

Description for `schema generate sobject`

This command is interactive and must be run in a Salesforce DX project directory. You're required to specify the object's label with the "--label" flag. The command uses this label to provide intelligent suggestions for other object properties, such as its API name and plural label.

All Salesforce objects are required to have a Name field, so this command also prompts you for the label and type of the Name field. Run the "sf metadata generate field" command to create additional fields for the object.

To reduce the number of prompts, use the "--use-default-features" flag to automatically enable some features, such as reporting and search on the object.

Examples for `schema generate sobject`

Create a custom object with the specified label and be prompted for additional information:

```
sf schema generate sobject --label "My Object"
```

Create a custom object and enable optional features without prompting:

```
sf schema generate sobject --label "My Object" --use-default-features
```

Flags

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-l | --label LABEL
Required

The custom object's label.

Type: option

-f | --use-default-features
Optional

Enable all optional features without prompting.

Enables these features:

- Search: Allows users to find the custom object's records when they search, including SOSL.

- Feeds: Enables feed tracking.

- Reports: Allows reporting of the data in the custom object records.

- History: Enables object history tracking.

- Activities: Allows users to associate tasks and scheduled calendar events related to the custom object records.

- Bulk API: With Sharing and Streaming API, classifies the custom object as an Enterprise Application object.

- Sharing: With Bulk API and Streaming API, classifies the custom object as an Enterprise Application object.

- Streaming API: With Bulk API and Sharing, classifies the custom object as an Enterprise Application object.

Type: boolean

Aliases for `schema generate sobject`

```
generate:metadata:sobject
```

`schema generate tab`

Generate the metadata source files for a new custom tab on a custom object.

Description for `schema generate tab`

Custom tabs let you display custom object data or other web content in Salesforce. Custom tabs appear in Salesforce as an item in the app's navigation bar and in the App Launcher.

This command must be run in a Salesforce DX project directory. You must pass all required information to it with the required flags. The source files for the custom object for which you're generating a tab don't need to exist in your local project.

Examples for `schema generate tab`

Create a tab on the `MyObject__c` custom object:

```
sf schema generate tab --object `MyObject__c` --icon 54 --directory
force-app/main/default/tabs
```

Flags

`--json`

Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`

Optional

Import flag values from a directory.

Type: option

-o | --object OBJECT
Required

API name of the custom object you're generating a tab for.

The API name for a custom object always ends in `__c`, such as `MyObject__c`.

Type: option

-d | --directory DIRECTORY
Required

Path to a "tabs" directory that will contain the source files for your new tab.

Type: option

-i | --icon ICON
Required

Number from 1 to 100 that specifies the color scheme and icon for the custom tab.

See https://lightningdesignsystem.com/icons/#custom for the available icons.

Type: option

Default value: 1

Aliases for `schema generate tab`

```
generate:metadata:tab
```

`sobject` Commands

Commands to interact with Salesforce objects.

sobject describe
Display the metadata for a standard or custom object or a Tooling API object.

sobject list
List all Salesforce objects of a specified category.

`sobject describe`

Display the metadata for a standard or custom object or a Tooling API object.

Description for `sobject describe`

The metadata is displayed in JSON format. See this topic for a description of each property: https://developer.salesforce.com/docs/atlas.en-us.api.meta/api/sforce_api_calls_describesobjects_describesobjectresult.htm.

This command displays metadata for Salesforce objects by default. Use the --use-tooling-api flag to view metadata for a Tooling API object.

Examples for `sobject describe`

Display the metadata of the "Account" standard object in your default org:

```
sf sobject describe --sobject Account
```

Display the metadata of the "MyObject__c" custom object in the org with alias "my-scratch-org":

```
sf sobject describe --sobject MyObject__c --target-org my-scratch-org
```

Display the metadata of the ApexCodeCoverage Tooling API object in your default org:

```
sf sobject describe --sobject ApexCodeCoverage --use-tooling-api
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-s | --sobject SOBJECT`
Required

API name of the object to describe.

Type: option

`-t | --use-tooling-api`
Optional

Use Tooling API to display metadata for Tooling API objects.

Type: boolean

Aliases for `sobject describe`

```
force:schema:sobject:describe
```

`sobject list`

List all Salesforce objects of a specified category.

Description for `sobject list`

You can list the standard objects, custom objects, or all. The lists include only Salesforce objects, not Tooling API objects.

Examples for `sobject list`

List all objects in your default org:

```
sf sobject list --sobject all
```

List only custom objects in the org with alias "my-scratch-org":

```
sf sobject list --sobject custom --target-org my-scratch-org
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-o | --target-org TARGET-ORG`
Required

Username or alias of the target org. Not required if the `target-org` configuration variable is already set.

Type: option

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-s | --sobject SOBJECT`
Optional

Category of objects to list.

Type: option

Default value: ALL

Aliases for `sobject list`

```
force:schema:sobject:list
```

`static-resource` Commands

Work with static resources.

`static-resource generate`

Generate a static resource.

Description for `static-resource generate`

Generates the metadata resource file in the specified directory or the current working directory. Static resource files must be contained in a parent directory called "staticresources" in your package directory. Either run this command from an existing directory of this name, or use the --output-dir flag to create one or point to an existing one.

Examples for `static-resource generate`

Generate the metadata file for a static resource called MyResource in the current directory:

```
sf static-resource generate --name MyResource
```

Similar to previous example, but specifies a MIME type of application/json:

```
sf static-resource generate --name MyResource --type application/json
```

Generate the resource file in the "force-app/main/default/staticresources" directory:

```
sf static-resource generate --name MyResource --output-dir
force-app/main/default/staticresources
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated static resource.

This name can contain only underscores and alphanumeric characters, and must be unique in your org. It must begin with a letter, not include spaces, not end with an underscore, and not contain two consecutive underscores.

Type: option

--type TYPE

Optional

Content type (mime type) of the generated static resource.

Must be a valid MIME type such as application/json, application/javascript, application/zip, text/plain, text/css, etc.

Type: option

Default value: application/zip

-d | --output-dir OUTPUT-DIR

Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

--api-version API-VERSION

Optional

Override the api version used for api requests made by this command

Type: option

Aliases for `static-resource generate`

```
force:staticresource:create
```

`visualforce` Commands

Work with Visualforce components.

visualforce generate component

Generate a Visualforce Component.

visualforce generate page

Generate a Visualforce Page.

`visualforce generate component`

Generate a Visualforce Component.

Description for `visualforce generate component`

The command generates the .Component file and associated metadata file in the specified directory or the current working directory by default.

Examples for `visualforce generate component`

Generate the metadata files for a Visualforce component in the current directory:

```
sf visualforce generate component --name mycomponent --label mylabel
```

Similar to previous example, but generate the files in the directory "force-app/main/default/components":

```
sf visualforce generate component --name mycomponent --label mylabel --output-dir components
```

Flags

`--json`
Optional

Format output as json.

Type: boolean

`--flags-dir FLAGS-DIR`
Optional

Import flag values from a directory.

Type: option

`-n | --name NAME`
Required

Name of the generated Visualforce Component.

The name can be up to 40 characters and must start with a letter.

Type: option

`-t | --template TEMPLATE`
Optional

Template to use for file creation.

Supplied parameter values or default values are filled into a copy of the template.

Type: option

Permissible values are: DefaultVFComponent

Default value: DefaultVFComponent

`-d | --output-dir OUTPUT-DIR`
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

-l | --label LABEL
Required

Visualforce Component label.

Type: option

Aliases for `visualforce generate component`

```
force:visualforce:component:create
```

`visualforce generate page`

Generate a Visualforce Page.

Description for `visualforce generate page`

The command generates the .Page file and associated metadata file in the specified directory or the current working directory by default.

Examples for `visualforce generate page`

Generate the metadata files for a Visualforce page in the current directory:

```
sf visualforce generate page --name mypage --label mylabel
```

Similar to previous example, but generate the files in the directory "force-app/main/default/pages":

```
sf visualforce generate page --name mypage --label mylabel --output-dir pages
```

Flags

--json
Optional

Format output as json.

Type: boolean

--flags-dir FLAGS-DIR
Optional

Import flag values from a directory.

Type: option

-n | --name NAME
Required

Name of the generated Visualforce Page.

The name can be up to 40 characters and must start with a letter.

Type: option

-d | --output-dir OUTPUT-DIR
Optional

Directory for saving the created files.

The location can be an absolute path or relative to the current working directory. The default is the current directory.

Type: option

Default value: .

`--api-version API-VERSION`
Optional

Override the api version used for api requests made by this command

Type: option

`-l | --label LABEL`
Required

Visualforce Page label.

Type: option

Aliases for `visualforce generate page`

```
force:visualforce:page:create
```

Help for `sf` Commands

The `-h` and `--help` flags show details about `sf` topics and their commands.

The short `-h` flag shows a subset of the command-line help and is meant for quick reference. The long `--help` flag shows the complete command-line help.

The short help (`-h`) for commands has these parts.

1. **Short Description of Command**

   At the top of the `-h` output (with no heading), a short description of the command is shown.

2. **Usage**

   The command signature on the Usage line uses the docopt format.

   - All available flags are listed. Flags that have short names are listed using their short names.
   - Flags that take a value show `<value>` immediately after the flag's name.
   - Optional flags are in square brackets (`[ ... ]`).
   - Required flags have no annotation.
   - For flags that accept a limited set of values, the values are shown after the flag name, separated by pipes (`--flagname value1|value2|value3`).

3. **Flags**

   The Flags section lists all the command's flags, including their short name, long name, and purpose. Flags are grouped for easier reading, such as global flags and other groups relevant to a specific command.

   For flags that take multiple values, you have two ways to specify the values:

   - Specify the flag multiple times, where each flag takes a different single value.
   - Specify the flag one time, but separate all the values with a space.

For example, the following commands are equivalent:

```
sf deploy metadata --metadata ApexClass --metadata CustomObject --metadata
AnotherCustomObject
sf deploy metadata --metadata ApexClass CustomObject AnotherCustomObject
```

Flags that accept a limited list of values include the values in parentheses, with the default value specified with the `default` keyword.

The long help (`--help`) for commands has the same parts as the preceding short help `-h`) and these additional parts.

1. **Description**

   Usage notes.

2. **Examples**

   All examples include a brief description.

3. **Flag Descriptions**

   Some flags have optional additional usage notes.

Salesforce Functions (Retired)

Salesforce Functions was retired on Jan 31, 2025. We removed the reference information for the Salesforce Functions CLI commands from this guide on Feb 5, 2025.

If you need reference information about the Salesforce Functions commands, see the Winter '25 Salesforce CLI Command Reference.

sfdx (Deprecated)

The `sfdx`-style commands are deprecated. We removed the reference information for them on June 12, 2024.

As of April 2023, maintenance of `sfdx` (v7) has ended, and we no longer support it. We strongly discourage using `sfdx` (v7) due to the absence of any updates, bug fixes, or technical support. We urge you to transition to the latest version of Salesforce CLI, `sf` (v2), to ensure compliance with current security standards and to benefit from ongoing enhancements. Failure to upgrade can result in unresolvable issues.

If you need reference information about the `sfdx`-style commands, see the Spring '24 Salesforce CLI Command Reference.

Migrate `sfdx`-Style Commands to Their `sf`-Style Equivalents

We recommend that you start using the new `sf` commands in your continuous integration (CI) scripts and in your day-to-day work as soon as possible. This section provides information on how to migrate. Migrating scripts are the focus, though this section also applies to running impromptu commands at a terminal. The `sfdx`-style commands, such as `force:org:create`, continue to work for now, although they're deprecated.

Follow these high-level steps to migrate.

- If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)) immediately. See Move from `sfdx` (v7) to `sf` (v2) for details.
- Read an overview of the changes, including important usage differences.
- Run `dev convert script` to convert a large portion of your CI script.
- Manually update the remainder of your script.

Where is the Reference for `sfdx`-Style Commands?

We removed the reference information about `sfdx`-style commands from this guide on June 12, 2024. If you need the information, see the Spring '24 Salesforce CLI Command Reference.

Overview of Command and Usage Differences

The deprecated `sfdx` commands are different from the `sf` ones in these key ways. Other sections in this migration guide go into details.

- Some commands and flags have new names, but their behavior and JSON output is the same as their `sfdx` equivalents. To migrate, rename your existing commands and flags. For example, let's say you have this command.

  ```
  force:apex:execute --targetusername <org> ---apexcodefile <file>
  ```

  Here's its `sf`-style equivalent.

  ```
  apex run --target-org <org> --file <file>
  ```

- For other `sfdx`-style commands, we created `sf` commands that likely behave differently, so migrating to them requires a bit more effort. The inputs and JSON output for these new commands are also likely different from their `sfdx` equivalents. For example, `force:org:create` is now two commands: `org create scratch` and `org create sandbox`.

  In these cases the `sfdx` command is still available for backward compatibility.

- Configuration and environment variables have new names. For example, `targetusername` is now `target-org`, and SFDX_DEFAULTUSERNAME is now SF_TARGET_ORG.
- We no longer use the `force` topic, except for a handful of commands that we kept for backward compatibility.

Read these usage differences between the `sfdx`-style and `sf`-style commands, and apply them when necessary.

- When flags for new `sf` commands take multiple values, you specify the flag multiple times, with each flag taking a different single value. For example:

  ```
  sf project deploy start --metadata ApexClass:SampleDataController --metadata
  ApexClass:PropertyController
  ```

  Previously, with the deprecated `sfdx`-style commands, you specified the flag one time and separated the values with commas. For example:

  ```
  sfdx force:source:deploy --metadata
  ApexClass:SampleDataController,ApexClass:PropertyController
  ```

  You can continue using this comma-separated style with existing commands before you migrate. But when you migrate to the `sf` commands, make sure that you use this new style because new and future commands don't support the comma-separated style. An example is specifying multiple Apex test classes and code coverage formats to the `project deploy start` command. If you continue using commas, the command doesn't return an error, but the Apex tests probably didn't all run.

For example, use this syntax.

```
sf project deploy start --metadata ApexClass --tests FirstTest --tests SecondTest --tests
 "Third Test" --coverage-formatters json --coverage-formatters html
```

But don't use this syntax.

```
sf project deploy start --metadata ApexClass --tests FirstTest,SecondTest,"Third Test"
 --coverage-formatters json,html
```

- The `sf` commands accept either spaces or colons between topics, commands, and subcommands. For example, both of these command formats to get a configuration variable are valid.

```
sf config get target-org
sf config:get target-org
```

Run the `dev convert script` Command

Begin your migration by running the `dev convert script` command to update your CI scripts. The command replaces many of the `sfdx` commands and flags with their `sf` equivalents.

Warning: We provide the `dev convert script` command to get you started with the migration. To ensure that they work as expected, you must test the converted scripts thoroughly.

First, install the `plugin-dev` Salesforce CLI plugin, which contains the conversion command.

```
sfdx plugins install @salesforce/plugin-dev
```

Then pass your script file to the `dev convert script` command with the `--script` flag.

```
sfdx dev convert script --script ./myScript.yml
```

The command scans your script file, and each time it finds an `sfdx` command or flag, it prompts whether you want to replace it with the new `sf` equivalent. The command overwrites your original file.

While `dev convert script` can convert a large portion of your script, it likely can't convert all of it because there's not always a one-to-one mapping between the previous and new commands. In these cases, `dev convert script` doesn't replace the `sfdx`-style command but instead adds a comment that starts with `#ERROR`.

Migrate Scripts Manually

Because `dev convert script` typically can't convert your entire script, you must migrate the remainder of the commands manually. You can update your entire script manually if you want.

The easiest way to find the `sf`-style equivalent of a `sfdx` command is to read the `sfdx` section of the Salesforce CLI Command Reference. Each deprecated `sfdx` command displays information about the new equivalent `sf` command and the new flag names.

You can also look at the deprecation warnings when you run an `sfdx` command. The warnings display the new `sf`-style equivalent command and flag names. To display help information about the *new* equivalent command along with examples, run the old command with the `--help` flag.

Most commands are a simple one-to-one mapping, including flag name changes. Let's take `auth:jwt:grant` as an example. The reference tells you to use the new `org login jwt` command instead, and it lists how the flag names have changed. Here's an example of the deprecated `sfdx`-style command.

```
sfdx auth:jwt:grant --username jdoe@example.org --jwtkeyfile /Users/jdoe/JWT/server.key
--clientid 123456 --setdefaultdevhubusername
```

Here's the `sf`-style equivalent.

```
sf org login jwt --username jdoe@example.org --jwt-key-file /Users/jdoe/JWT/server.key
--client-id 123456 --set-default-dev-hub
```

The `force:apex` commands also have a one-to-one mapping to the new `sf`-style commands. Here's an example of the `force:apex:test:run` command.

```
sfdx force:apex:test:run --suitenames "MySuite,MyOtherSuite" --codecoverage
--detailedcoverage --targetusename my-scratch --outputdir tests/output"
```

Here's the `sf`-style equivalent.

```
sf apex run test --suite-names "MySuite,MyOtherSuite" --code-coverage --detailed-coverage
 --target-org my-scratch --output-dir tests/output"
```

Some commands aren't a direct one-to-one mapping, or their behavior changed, so migrating them requires more effort. For additional information about these commands, see these topics.

- force:source:* and force:mdapi:* Commands Migration
- force:org:* Commands Migration
- force:data:bulk:* Commands Migration

High-Level Overview of Common Flag Name Changes

This table provides an overview of common Salesforce CLI flag name changes.

`sfdx`-Style Flag Name	`sf`-Style Flag Name
`--targetusername`, `-u`	`--target-org`, `-o`
`--targetdevhubusername`, `-v`	`--target-dev-hub`, `-v`
`--apiversion`	`--api-version`
`--loglevel`	No equivalent. Use the SF_LOG_LEVEL environment variable instead.
`--json`	`--json` (No change)

For less common flags, the `sf`-style name is often similar to the `sfdx`-style one, but it has dashes to make it easier to read. We also standardized many of the flags across all topics and commands, such as using `--output-dir` consistently for the directory to write the results of a command. Here are a few more examples.

- `project:create --outputdir` is now `project generate --output-dir`.
- `force:source:deploy --sourcepath` is now `project deploy start --source-dir`.
- `force:apex:class:create --classname` is now `apex generate class --name`.

- `force:package:create --errornotificationusername` is now `package create --error-notification-username`.

As always, for command and flag name changes for a specific deprecated `sfdx` command, see its reference page in the Salesforce CLI Command Reference.

`force:source:*` and `force:mdapi:*` Commands Migration

Migrating the `force:source:*` and `force:mdapi:*` commands is straightforward in most cases, although some scenarios require some rework.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

We introduced two `sf`-style commands, `project deploy start` and `project retrieve start`, to replace these six deprecated `force` commands.

- `force:source:push`
- `force:source:pull`
- `force:source:deploy`
- `force:source:retrieve`
- `force:mdapi:deploy`
- `force:mdapi:retrieve`

It was often confusing to determine which `force` command to use because they all have similar functionality. For example, both `force:source:push` and `force:source:deploy` move source format files from your project to the org. Now it's simple: use `project deploy start` to deploy metadata to your org and `project retrieve start` to retrieve metadata from your org.

By default, both new commands work with files in source format. If you want to deploy or retrieve in metadata format, use the `--metadata-dir` or `--target-metadata-dir` flags, respectively.

The `project deploy|retrieve start` commands support source tracking. However, because these two commands encapsulate the functionality of the six `force` commands, source tracking works a bit differently. For more information, see Source Tracking in New sf-Style Commands.

The table summarizes the mapping between the `force:source:*` and `force:mdapi:*` commands to their new `sf`-style equivalents. The usage notes indicate if the mapping is a simple one-to-one. If it is, you migrate them by replacing their command and flag names in your scripts. Some command migrations require changes beyond simple name replacements, or the functionality has changed, as described in the usage notes. For more guidance, see the examples after the table.

For all command migrations, refer to the reference pages for each `force` command in the Salesforce CLI Command Reference for details.

- `force:source:*`
- `force:mdapi:*`

`force:source|mdapi:*` **Command**	**Equivalent `sf`-Style Command**	**Usage Notes**
`force:mdapi:convert`	`project convert mdapi`	One-to-one mapping
`force:mdapi:deploy`	`project deploy start`	The `project deploy start` command works with source and metadata format files. The default is source format. To deploy metadata format files, use the `--metadata-dir` flag.
`force:mdapi:deploy:cancel`	`project deploy cancel`	One-to-one mapping
`force:mdapi:deploy:report`	`project deploy report` or `project deploy resume`	The `force:mdapi:deploy:report` command does more than just report, it also resumes a deployment. We now provide two new commands for each task, which is more intuitive. The `project deploy resume` command doesn't include the `--wait -1` flag, which means "wait forever."Instead, specify a large number with the new commands to ensure enough time to complete the deployment.
`force:mdapi:describemetadata`	`org list metadata-types`	One-to-one mapping. The new command is in the `org` topic.
`force:mdapi:listmetadata`	`org list metadata`	One-to-one mapping. The new command is in the `org` topic.

`force:source|mdapi:*` Command	Equivalent `sf`-Style Command	Usage Notes
`force:mdapi:retrieve`	`project retrieve start`	The `project retrieve start` command works with source and metadata format files. The default is source format. To retrieve metadata format files, use the `--target-metadata-dir` flag.
`force:mdapi:retrieve:report`	No equivalent.	We removed this command and provide no new equivalent because it's not needed.
`force:source:convert`	`project convert source`	One-to-one mapping
`force:source:delete`	`project delete source`	One-to-one mapping
`force:source:deploy`	`project deploy start`	When deploying in source format, the `project deploy start` command always tracks your source if the org is enabled for source tracking. So there's no new equivalent for the `force:source --tracksource` flag. If you don't want to use source tracking, you have two options: • Create an org that doesn't have source tracking enabled. • Run the `org disable tracking` command on an existing org. Instead of `force:source:deploy --validateddeployrequestid`, use the new `project deploy validate` and `project deploy quick --job-id` commands.
`force:source:deploy:cancel`	`project deploy cancel`	One-to-one mapping
`force:source:deploy:report`	`project deploy report` or `project deploy resume`	The `force:source:deploy:report` command does more than just report, it also resumes a deployment. We now provide two new commands for each task, which is more intuitive.
`force:source:ignored:list`	`project list ignored`	One-to-one mapping
`force:source:manifest:create`	`project generate manifest`	One-to-one mapping
`force:source:open`	`org open --source-file`	The `force:source:open` command is now the `--source-file` flag of the `org open` command. Also, you can

`force:source|mdapi:*` **Command**	**Equivalent `sf`-Style Command**	**Usage Notes**
		now specify a browser with the `--browser` flag.
`force:source:pull`	`project retrieve start`	The new `project retrieve start` command has additional flags that you can use to fine-tune the retrieve.
`force:source:push`	`project deploy start`	The `project deploy start` command doesn't support the `pushPackageDirectoriesSequentially` property of `sfdx-project.json`. The `force:source:push` command uses this property to deploy packages sequentially. If you must deploy packages sequentially, and in a specific order, use separate `project deploy start` commands in the desired order. The new `project deploy start` command has additional flags that you can use to fine-tune the deploy.
`force:source:retrieve`	`project retrieve start`	When retrieving in source format, the `project retrieve start` command always tracks your source if the org is enabled for source tracking. So there's no new equivalent for the `force:source --tracksource` flag. If you don't want to use source tracking, you have two options: • Create an org that doesn't have source tracking enabled. • Run the `org disable tracking` command on an existing org.
`force:source:status`	`project deploy preview` or `project retrieve preview`	We now provide two separate commands to preview a deploy or a retrieve, which is more intuitive. These `preview` commands have flags that align with their non-preview commands, such as `project deploy start`. The `force:source:status` command shows local and remote changes.
`force:source:tracking:clear`	`project delete tracking`	One-to-one mapping
`force:source:tracking:reset`	`project reset tracking`	One-to-one mapping

A few examples can help you get started with these new commands.

Note: To differentiate the examples, we preface `sfdx`-style commands with `sfdx` and `sf`-style commands with `sf`. However, you can indicate either `sf` or `sfdx` when running any CLI command.

`force:source` Examples

This `force` command converts source-formatted files into metadata format.

```
sfdx force:source:convert --rootdir path/to/source --outputdir path/to/outputdir
--packagename "My Package"
```

Here's the `sf`-style equivalent.

```
sf project convert source --root-dir path/to/source --output-dir path/to/outputdir
--package-name 'My Package'
```

This `force` command deploys multiple metadata types.

```
sfdx force:source:deploy --metadata "ApexClass,CustomObject" --testlevel RunSpecifiedTests
 --runtests MyTests --targetusername my-scratch
```

Here's the `sf`-style equivalent in which the `--metadata` flag is specified multiple times.

```
sf project deploy start --metadata ApexClass --metadata CustomObject --test-level
RunSpecifiedTests --tests MyTests --target-org my-scratch
```

This `force` command pushes (deploys) all the changes in your project to an org.

```
sfdx force:source:push --targetusername myscratch --forceoverwrite --wait 10
```

Here's the `sf`-style equivalent.

```
sf project deploy start --target-org myscratch --ignore-conflicts --wait 10
```

This `force` command retrieves the source in the specified directories.

```
sfdx force:source:retrieve --sourcepath
"path/to/objects/MyCustomObject/fields/MyField.field-meta.xml, path/to/apex/classes"
```

Here's the `sf`-style equivalent in which the `--source-dir` flag is specified multiple times.

```
sf project retrieve start --source-dir
path/to/objects/MyCustomObject/fields/MyField.field-meta.xml --source-dir
path/to/apex/classes
```

This `force` command opens a metadata file in Lightning App Builder.

```
sfdx force:source:open --source-file
force-app/main/default/flexipages/Hello.flexipage-meta.xml --urlonly --targetusername
myscratch
```

Here's the `sf`-style equivalent that uses the `org open` command.

```
sf org open --source-path force-app/main/default/flexipages/Hello.flexipage-meta.xml
--url-only --target-org myscratch
```

This `force` command pulls (retrieves) all the changes in your org to your project.

```
sfdx force:source:pull --targetusername myscratch --forceoverwrite --wait 10
```

Here's the `sf`-style equivalent.

```
sf project retrieve start --target-org myscratch --ignore-conflicts --wait 10
```

This `force` command shows how your local project differs from the org.

```
sfdx force:source:status --targetusername myscratch --local
```

Here's the `sf`-style equivalent; the command requires that you specify what you want preview, in this case, with the `--manifest` flag.

```
sf project deploy preview --target-org myscratch --manifest package.xml
```

`force:mdapi` Examples

This `force` command deploys metadata format files in the specified directory.

```
sfdx force:mdapi:deploy --deploydir some/path --wait 1000 --checkonly --testlevel
RunAllTestsInOrg --targetusername my-test-org
```

There are two `sf`-style equivalents.

```
sf project deploy start --dry-run --metadata-dir some/path --wait 1000 --test-level
RunAllTestsInOrg --target-org my-test-org
sf project deploy validate --metadata-dir some/path --wait 1000 --test-level RunAllTestsInOrg
 --target-org my-test-org
```

This `force` command deploys a .zip file that contains metadata files.

```
sfdx force:mdapi:deploy sfdx force:mdapi:deploy --zipfile stuff.zip --resultsdir --junit
```

Here's the `sf`-style equivalent.

```
sf project deploy start --metadata-dir stuff.zip --results-dir --junit
```

This `force` command retrieves metadata defined in a manifest file into the target directory.

```
sfdx force:mdapi:retrieve --retrievetargetdir path/to/retrieve/dir --unpackaged package.xml
```

Here's the `sf`-style equivalent.

```
sf project retrieve start --target-metadata-dir path/to/retrieve/dir --manifest package.xml
```

Overview of New Commands and Functionality

In addition to the new `project deploy start` and `project retrieve start`, we introduced other commands and flags that improve the Salesforce CLI's usability.

- Preview a deployment to your org with the `project deploy preview` command.

  The command outputs a table that shows what happens when you run the `project deploy start` command. The table displays a preview of the metadata components that are deployed and deleted, and the current conflicts between your project and org. The table also lists the files that aren't deployed because they're included in your `.forceignore` file.

- Similarly, preview a retrieve from your org with the `project retrieve preview` command.
- Validate a deployment, and then quickly deploy it later, with the `project deploy validate` and `project deploy quick` command pair.

Use `project deploy validate` to verify whether a deployment can succeed without actually deploying the metadata to your org. This command is similar to `project deploy start`, except that you're required to run Apex tests, and the command returns a job ID rather than actually executing the deployment. If the validation succeeds, then you pass this job ID to the `project deploy quick` command to actually deploy the metadata. This type of deploy takes less time because it skips running Apex tests.

You can also use the `--dry-run` flag of `project deploy start` to get a preview of a deploy. Use this preview method if you don't plan to later do a quick deploy. This way of previewing provides more flexibility because you can use all the flags of the `project deploy start` command, such as making destructive changes with the `--pre|post-destructive-changes` flags. The `project deploy validate` provides just a subset of the full deployment flags.

- These new deploy commands that take a job ID now also have the handy `--use-most-recent` flag to automatically use the job ID of the most recent deploy operation.
  - `project deploy cancel`
  - `project deploy quick`
  - `project deploy report`
  - `project deploy resume`
- These new deploy commands have the `--async` flag to run the command asynchronously.
  - `project deploy cancel`
  - `project deploy quick`
  - `project deploy resume`
  - `project deploy validate`
- Delete source from a non-source-tracked org with the `project delete source` command.

`force:org:*` Commands Migration

Migrating the `force:org:*` commands is straight forward in most cases, although some scenarios require some rework.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

This table summarizes the mapping between the existing `force:org:*` commands and their new `sf`-style equivalents. The usage notes indicate if the mapping is one-to-one. If it is, you migrate them by changing their command and flag names as listed in the reference page for each `force:org:*` command. Some command migrations require more changes, as described in the usage notes. For more guidance, see the examples after the table.

`force:org:*` Command	Equivalent `sf`-Style Command	Usage Notes
`force:org:clone`	`org create sandbox`	To clone a sandbox, specify the `--clone` flag of the `org create sandbox` command and set it to the name of the sandbox being cloned. To specify the new sandbox's name, use the `--name` flag t. These two flags replace the `SandboxName` and `SourceSandboxName` key value pairs

`force:org:*` Command	Equivalent `sf`-Style Command	Usage Notes
		that you could specify with the `force:org:clone` command. See examples later in this section.
`force:org:create`	`org create scratch` or `org create sandbox`	We provide two commands to create an org, one for sandbox and one for scratch org. You can no longer specify key-value pairs, such as `sandboxName=FullSbx`, from the scratch or sandbox configuration file at the command line. You must either use the provided flags or update the definition file. See examples later in this section.
`force:org:delete`	`org delete scratch` or `org delete sandbox`	We provide two commands to delete an org, one for sandbox and one for scratch org.
`force:org:display`	`org display`	One-to-one mapping
`force:org:list`	`org list`	One-to-one mapping
`force:org:open`	`org open`	One-to-one mapping
`force:org:status`	`org resume sandbox`	The `force:org:status` command did more than report the status of a sandbox create, it also resumed a sandbox create if it wasn't complete. We now provide an explicit command to resume the org creation, which is more intuitive. To specify the incomplete sandbox creation job, use the `--job-id` or `--use-most-recent` flags.

We also introduced the command `org resume scratch` to resume a scratch org creation if it times out. Previously, you could no longer connect to it, and you manually deleted it from your Dev Hub org. Now you can resume where it left off using a job ID or the `--use-most-recent` flag. When the org creation finishes, the command automatically authenticates to the org, saves the org info locally, and deploys any configured settings.

A few examples can help you get started with these new commands.

Note: To differentiate the examples, we preface `sfdx`-style commands with `sfdx` and `sf`-style commands with `sf`. However, you can indicate either `sf` or `sfdx` when running any CLI command.

Let's start with the deprecated `force:org:create` command. The reference tells you to use either `org create sandbox` or `org create scratch`, depending on what you want to create. Let's say you want to migrate this `force` command.

```
sfdx force:org:create --definitionfile config/scratch-def.json --setalias MyScratchOrg
--targetdevhubusername MyDevHub --nonamespace --setdefaultusername
```

Because the command creates a scratch org, use this equivalent `sf` command.

```
sf org create scratch --definition-file config/scratch-def.json --alias MyScratchOrg
--target-dev-hub MyDevHub --no-namespace --set-default
```

This `force` example specifies scratch org options as key-value pairs at the command line, which is no longer allowed.

```
sfdx force:org:create adminEmail=me@email.com edition=Developer
username=admin_user@orgname.org --country=GB --targetdevhubusername MyDevHub
```

In the `sf`-style equivalent, use the `--edition`, `--admin-email`, and `--username` flags instead. But because `country` doesn't have an equivalent flag, you must specify a scratch org definition file that contains the `country` option. Here's what the new command looks like.

```
sf force:org:create --definition-file config/scratch-def.json --admin-email me@email.com
--edition=developer --username=admin_user@orgname.org --targetdevhubusername MyDevHub
```

In the previous example, the `--edition` flag takes lowercase values for Salesforce editions. To see the full list of valid editions, run `org create scratch -h`.

Here's an example of a scratch org definition file that contains the `country` option.

```
{
    "orgName": "Dreamhouse",
    "edition": "Developer",
    "country": "GB",
    "features": ["Walkthroughs", "EnableSetPasswordInApi"],
    "settings": {
        "lightningExperienceSettings": {
            "enableS1DesktopEnabled": true
        },
        "mobileSettings": {
            "enableS1EncryptedStoragePref2": false
        }
    }
}
```

This `force` example creates a sandbox.

```
sfdx force:org:create --type sandbox --definitionfile config/dev-sandbox-def.json --setalias
 MyDevSandbox --targetusername ProdOrg
```

Here's the equivalent `sf`-style command.

```
sf org create sandbox --definition-file config/dev-sandbox-def.json --alias MyDevSandbox
--target-org ProdOrg
```

This `force` example clones a sandbox by specifying the `SourceSandboxName` and `SandboxName` key-value pairs at the command line.

```
sfdx force:org:clone --type sandbox SourceSandboxName=ExistingSandbox
SandboxName=NewClonedSandbox --setalias MyDevSandbox --targetusername ProdOrg
```

In the `sf`-style command, use flags instead.

```
sf org create sandbox --clone ExistingSandbox --name NewClonedSandbox --alias MyDevSandbox
 --target-org ProdOrg
```

This `force` example deletes a scratch org.

```
sfdx force:org:delete --targetusername MyScratchOrg --noprompt
```

Here's the equivalent `sf`-style command.

```
sf org delete scratch --target-org MyScratchOrg --no-prompt
```

`force:data:bulk:*` Commands Migration

We added four new `sf` commands that use Bulk API 2.0 to upsert and delete data to and from your org. All the `sfdx` commands use Bulk API 1.0.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

These new `sf` commands use Bulk API 2.0.

- `data delete bulk`
- `data delete resume`
- `data upsert bulk`
- `data upsert resume`

We generally recommend that you start using the new `sf` commands instead of these equivalent `sfdx` commands that use Bulk API 1.0.

- `force:data:bulk:delete`
- `force:data:bulk:upsert`
- `force:data:bulk:status`

However, one reason to keep using the `force:data:bulk:upsert` command is if you want to run the upsert serially with the `--serial` flag. The new Bulk API 2.0 commands don't support serial execution. For this reason, and for users who want to continue using Bulk API 1.0, we aren't deprecating the `force:data:bulk:*` commands at this time.

Configuration and Environment Variable Names Migration

Because the `dev convert script` conversion command doesn't update configuration and environment variables to their new names, we recommend that you update them manually to avoid deprecation warnings. Although the existing `sfdx`-style variable names continue to work, we recommend that you start using the new `sf`-style ones. When you use the old ones, you get a warning with the name of the new configuration and environment variable to use.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

Configuration Variables

The `sfdx`-style configuration variables are aliased to their `sf`-style equivalents. As a result, you can use either the `sfdx` or the `sf` variable names with the `config` commands. But the commands always work on the `sf` variable names. For example, `config set` and `config unset` always set the configuration with the `sf` name, even if you specify the `sfdx` name. All `config` commands display the `sf` name in their outputs, even if you specified the `sfdx` name in the command.

These examples show the rules in action.

```
sf config set defaultusername=my-scratch-org
Warning: Deprecated config name: defaultusername. Please use target-org instead.
Set Config
=================================
| Name       Value          Success
| ──────────── ──────────────── ───────
| target-org my-scratch-org true

sf config list
List Config
=========================================
| Name              Value          Location
| ──────────────────── ──────────────── ────────
| target-org        my-scratch-org Local

sf config get defaultusername
Warning: Deprecated config name: defaultusername. Please use target-org instead.
Get Config
=================================
| Name       Value          Success
| ──────────── ──────────────── ───────
| target-org my-scratch-org true
```

Use this table to migrate your scripts to use the new `sf`-style configuration variable names.

`sfdx`-style Configuration Value	Equivalent `sf`-style Configuration Variable
`apiVersion`	`org-api-version`
`customOrgMetadataTemplates`	`org-custom-metadata-templates`
`defaultdevhubusername`	`target-dev-hub`
`defaultusername`	`target-org`
`disableTelemetry`	`disable-telemetry`
`instanceUrl`	`org-instance-url`
`maxQueryLimit`	`org-max-query-limit`
`restDeploy`	`org-metadata-rest-deploy`

Environment Variables

You can set both the new `sf`-style and old `sfdx`-style environment variables. However, if they're set to different values, Salesforce CLI uses the `sf` one and displays a warning.

To migrate most environment variables, change the initial SFDX to SF. However, some variables have bigger changes, as displayed in this table, while others haven't changed their name. For the full list, see Salesforce CLI Environment Variables.

Equivalent `sf`-style Environment Variable	Equivalent `sf`-style Environment Variable
SFDX_API_VERSION	SF_ORG_API_VERSION

Equivalent `sf`-style Environment Variable	Equivalent `sf`-style Environment Variable
SFDX_CUSTOM_ORG_METADATA_TEMPLATES	SF_ORG_CUSTOM_METADATA_TEMPLATES
SFDX_DEFAULTDEVHUBUSERNAME	SF_TARGET_DEV_HUB
SFDX_DEFAULTUSERNAME	SF_TARGET_ORG
SFDX_INSTANCE_URL	SF_ORG_INSTANCE_URL
SFDX_MAX_QUERY_LIMIT	SF_ORG_MAX_QUERY_LIMIT
SFDX_REST_DEPLOY	SF_ORG_METADATA_REST_DEPLOY

For example, here's how to set your default Dev Hub org to an alias with an environment variable before running the command to create a scratch org.

```
SF_TARGET_DEV_HUB=MyDevHub sf org create scratch --definition-file config/scratch-def.json
```

Source Tracking in New `sf`-Style Commands

Source tracking in the new `sf`-style commands works basically the same as in the `sfdx`-style commands, but with a few small differences outlined in this topic.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

These `sf`-style commands support source tracking.

- `project deploy start`
- `project delete source`
- `project retrieve start`

The `sf`-style commands encapsulate the functionality of these six deprecated `sfdx`-style commands.

- `force:source:push`
- `force:source:pull`
- `force:source:deploy`
- `force:source:retrieve`
- `force:mdapi:deploy`
- `force:mdapi:retrieve`

Let's start with deploying. The first time you run `project deploy start` on a scratch or sandbox org that allows source tracking, the command deploys all the source files from your local project. But when you next run the command, it deploys only the files that changed locally. If you use one of the flags to narrow the deploy list, `--source-dir`, `--metadata`, or `--manifest`, then the command deploys only the changed files in the specified directory, metadata, or manifest. If you don't specify any of the flags, then the command deploys all changes in the project, similar to how the `sfdx`-style command `force:source:push` works.

If you run `project retrieve start` on a newly created org, nothing happens because there are no changes to track yet. When you next run the command, any changes in the org are retrieved. These changes include updates from other users who connect to the org, not just your changes. If you don't specify `--source-dir`, `--metadata`, or `--manifest`, then all changes in the org are retrieved, just like the `sfdx`-style command `force:source:pull`.

If one of these commands detects a conflict in the files you're about to deploy or retrieve, the command displays the conflicts. To force the deployment or retrieval of the changes, use the `--ignore-conflicts` flag. This flag is similar to the `--forceoverwrite` flag of many of the `force:source` commands. For example:

```
sf project deploy start --source-dir force-app --ignore-conflicts
```

Determine If Your Org Allows Source Tracking

Source tracking works only if your target org allows it. Don't worry, you can still deploy or retrieve metadata to and from an org without source tracking. But the commands don't check for conflicts, and you must specify what you want to deploy or retrieve using an appropriate flag, such as `--source-dir`.

Here's how to determine whether your org allows source tracking.

- For Developer Edition orgs, production orgs, Partial Copy sandboxes, and Full sandboxes, source tracking isn't available.
- For Developer and Developer Pro sandboxes:
  - Source tracking is enabled if their associated production org has been enabled for source tracking.
  - Source tracking is possible when you create the sandbox with the `--no-track-source` flag of the `org create sandbox` command. For example:

    ```
    sf org create sandbox --definition-file config/dev-sandbox-def.json --target-org
    prodOrg --no-track-source
    ```

- Scratch orgs have source tracking by default.
  - You can opt out of source tracking when you create the scratch org with the `--no-track-source` flag of the `org create scratch` command. This flag affects only your local configuration, not the scratch org itself. Salesforce CLI sets a local configuration option `trackSource: false` as part of your authorization information to the org. If you log out of the scratch org and then log back in again, source tracking is enabled again by default. Here's how to create a scratch org with source tracking disabled.

    ```
     sf org create scratch --target-dev-hub=MyHub --definition-file
    config/project-scratch-def.json --no-track-source
    ```

Tip: You can use the `org disable tracking` or `org enable tracking` commands to disable or enable source tracking on an existing org.

SEE ALSO:

Salesforce DX Developer Guide: Track Changes Between Your Project and Org

Mapping `sfdx` Commands to Their `sf` Equivalents

This table maps the `sfdx`-style commands, such as `force:org:create`, to their closest `sf`-style equivalent, such as `org create sandbox` or `org create scratch`. To help you migrate your continuous integration (CI) scripts to use the new `sf`-style commands, each `sfdx`-style entry links to a command reference page that provides more information.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

`sfdx`-style Command	Equivalent `sf`-style Command
`alias:list`	`alias list`
`alias:set`	`alias set`
`alias:unset`	`alias unset`
`auth:accesstoken:store`	`org login access-token`
`auth:device:login`	`org login device`
`auth:jwt:grant`	`org login jwt`
`auth:list`	`org list auth`
`auth:logout`	`org logout`
`auth:sfdxurl:store`	`org login sfdx-url`
`auth:web:login`	`org login web`
`autocomplete`	`autocomplete`
`config:get`	`config get`
`config:list`	`config list`
`config:set`	`config set`
`config:unset`	`config unset`
`doctor`	`doctor`
`force:analytics:template:create`	`analytics generate template`
`force:apex:class:create`	`apex generate class`
`force:apex:execute`	`apex run`
`force:apex:log:get`	`apex get log`
`force:apex:log:list`	`apex list log`
`force:apex:log:tail`	`apex tail log`
`force:apex:test:report`	`apex get test`
`force:apex:test:run`	`apex run test`
`force:apex:trigger:create`	`apex generate trigger`
`force:cmdt:create`	`cmdt generate object`
`force:cmdt:field:create`	`cmdt generate field`
`force:cmdt:generate`	`cmdt generate fromorg`
`force:cmdt:record:create`	`cmdt generate record`
`force:cmdt:record:insert`	`cmdt generate records`

`sfdx`-style Command	Equivalent `sf`-style Command
`force:community_create`	`community create`
`force:community:publish`	`community publish`
`force:community:template:list`	`community list template`
`force:data:bulk:delete`	• `data delete bulk` (Bulk API 2.0) • `force data bulk delete` (Bulk API 1.0)
`force:data:bulk:status`	• `data delete resume` (Bulk API 2.0) • `data upsert resume` (Bulk API 2.0) • `force data bulk status` (Bulk API 1.0)
`force:data:bulk:upsert`	• `data upsert bulk` (Bulk API 2.0) • `force data bulk upsert` (Bulk API 1.0)
`force:data:record:create`	`data create record`
`force:data:record:delete`	`data delete record`
`force:data:record:get`	`data get record`
`force:data:record:update`	`data update record`
`force:data:soql:bulk:report`	`data query resume`
`force:data:soql:query`	`data query`
`force:data:tree:export`	`data export tree`
`force:data:tree:import`	`data import tree`
`force:lightning:app:create`	`lightning generate app`
`force:lightning:component:create`	`lightning generate component`
`force:lightning:event:create`	`lightning generate event`
`force:lightning:interface:create`	`lightning generate interface`
`force:lightning:lwc:test:create`	`force lightning lwc test create`
`force:lightning:lwc:test:run`	`force lightning lwc test run`
`force:lightning:lwc:test:setup`	`force lightning lwc test setup`
`force:lightning:test:create`	`lightning generate test`
`force:limits:api:display`	`org list limits`
`force:limits:recordcounts:display`	`org list sobject record-counts`
`force:mdapi:convert`	`project convert mdapi`

`sfdx`-style Command	Equivalent `sf`-style Command
`force:mdapi:deploy`	`project deploy start --metadata-dir`
`force:mdapi:deploy:cancel`	`project deploy cancel`
`force:mdapi:deploy:report`	`project deploy report|resume`
`force:mdapi:describemetadata`	`org list metadata-types`
`force:mdapi:listmetadata`	`org list metadata`
`force:mdapi:retrieve`	`project retrieve start --target-metadata-dir`
`force:mdapi:retrieve:report`	No equivalent.
`force:org:clone`	`org create sandbox --clone`
`force:org:create`	• `org create scratch` • `org create sandbox`
`force:org:delete`	• `org delete scratch` • `org delete sandbox`
`force:org:display`	`org display`
`force:org:list`	`org list`
`force:org:open`	`org open`
`force:org:shape:create`	`org create shape`
`force:org:shape:delete`	`org delete shape`
`force:org:shape:list`	`org list shape`
`force:org:snapshot:create`	`org create snapshot`
`force:org:snapshot:delete`	`org delete snapshot`
`force:org:snapshot:get`	`org get snapshot`
`force:org:snapshot:list`	`org list snapshot`
`force:org:status`	`org resume sandbox`
`force:package1:version:create`	`package1 version create`
`force:package1:version:create:get`	`package1 version create get`
`force:package1:version:display`	`package1 version display`
`force:package1:version:list`	`package1 version list`
`force:package:create`	`package create`
`force:package:delete`	`package delete`

`sfdx`-style Command	Equivalent `sf`-style Command
force:package_install	package install
force:package:install:report	package install report
force:package:installed:list	package installed list
force:package:list	package list
force:package:uninstall	package uninstall
force:package:uninstall:report	package uninstall report
force:package:update	package update
force:package:version:create	package version create
force:package:version:create:list	package version create list
force:package:version:create:report	package version create report
force:package:version:delete	package version delete
force:package:version:displayancestry	package version displayancestry
force:package:version:list	package version list
force:package:version:promote	package version promote
force:package:version:report	package version report
force:package:version:update	package version update
force:project:create	project generate
force:schema:sobject:describe	sobject describe
force:schema:sobject:list	sobject list
force:source:convert	project convert source
force:source:delete	project delete source
force:source:deploy	project deploy start
force:source:deploy:cancel	project deploy cancel
force:source:deploy:report	project deploy report|resume
force:source:ignored:list	project list ignored
force:source:manifest:create	project generate manifest
force:source:open	org open --source-file
force:source:pull	project retrieve start
force:source:push	project deploy start
force:source:retrieve	project retrieve start

`sfdx`-style Command	Equivalent `sf`-style Command
`force:source:status`	• `project deploy preview` • `project retrieve preview`
`force:source:tracking:clear`	`project delete tracking`
`force:source:tracking:reset`	`project reset tracking`
`force:staticresource:create`	`static-resource generate`
`force:user:create`	`org create user`
`force:user:display`	`org display user`
`force:user:list`	`org list users`
`force:user:password:generate`	`org generate password`
`force:user:permset:assign`	`org assign permset`
`force:user:permsetlicense:assign`	`org assign permsetlicense`
`force:visualforce:component:create`	`visualforce generate component`
`force:visualforce:page:create`	`visualforce generate page`
`help`	`help`
`info:releasenotes:display`	`info releasenotes display`
`search`	`search`
`which`	`which`

Mapping `sf` Commands to Their `sfdx` Equivalents

This table maps the core `sf`-style commands, such as `org create sandbox`, to their closest `sfdx`-style equivalent, such as `force:org:create`.

Note: If you're still using version 7 of the `sfdx` executable of Salesforce CLI (also referred to as `sfdx` (v7)), move to version 2 of the `sf` executable (`sf` (v2)). See Move from `sfdx` (v7) to `sf` (v2) for details.

`sf`-style Command	Equivalent `sfdx`-style Command
`alias set`	`alias:set`
`alias unset`	`alias:unset`
`alias list`	`alias:list`
`analytics generate template`	`force:analytics:template:create`
`apex generate class`	`force:apex:class:create`
`apex generate trigger`	`force:apex:trigger:create`

`sf`-style Command	Equivalent `sfdx`-style Command
`apex run`	`force:apex:execute`
`apex get log`	`force:apex:log:get`
`apex list log`	`force:apex:log:list`
`apex tail log`	`force:apex:log:tail`
`apex get test`	`force:apex:test:report`
`apex run test`	`force:apex:test:run`
`autocomplete`	`autocomplete`
`cmdt generate object`	`force:cmdt:create`
`cmdt generate field`	`force:cmdt:field:create`
`cmdt generate fromorg`	`force:cmdt:generate`
`cmdt generate record`	`force:cmdt:record:create`
`cmdt generate records`	`force:cmdt:record:insert`
`community create`	`force:community:create`
`community list template`	`force:community:template:list`
`community publish`	`force:community:publish`
`config get`	`config:get`
`config list`	`config:list`
`config set`	`config:set`
`config unset`	`config:unset`
`data create record`	`force:data:record:create`
`data delete bulk` (Uses Bulk API 2.0)	`force:data:bulk:delete` (Uses Bulk API 1.0)
`data delete resume` (Uses Bulk API 2.0)	`force:data:bulk:status` (Uses Bulk API 1.0)
`data delete record`	`force:data:record:delete`
`data export tree`	`force:data:tree:export`
`data get record`	`force:data:record:get`
`data import tree`	`force:data:tree:import`
`data query`	`force:data:soql:query`
`data query resume`	`force:data:soql:bulk:report`
`data resume` (Deprecated)	`force:data:bulk:status`
`data update record`	`force:data:record:update`

`sf`-style Command	Equivalent `sfdx`-style Command
`data upsert bulk` (Uses Bulk API 2.0)	`force:data:bulk:upsert` (Uses Bulk API 1.0)
`data upsert resume` (Uses Bulk API 2.0)	`force:data:bulk:status` (Uses Bulk API 1.0)
`deploy` (interactive) Deprecated	No equivalent.
`doctor`	`doctor`
`help`	`help`
`info releasenotes display`	`info:releasenotes:display`
`lightning generate app`	`force:lightning:app:create`
`lightning generate component`	`force:lightning:component:create`
`lightning generate event`	`force:lightning:event:create`
`lightning generate interface`	`force:lightning:interface:create`
`lightning generate test`	`force:lightning:test:create`
`login` (interactive) Deprecated	No equivalent.
`logout` (interactive) Deprecated	No equivalent.
`org assign permset`	`force:user:permset:generate`
`org assign permsetlicense`	`force:user:permsetlicense:generate`
`org create shape`	`force:org:shape:create`
`org create snapshot`	`force:org:snapshot:create`
`org create user`	`force:user:create`
`org create sandbox`	`force:org:create --type sandbox`
`org create scratch`	`force:org:create --type scratch`
`org delete sandbox`	`force:org:delete`
`org delete scratch`	`force:org:delete`
`org delete shape`	`force:org:shape:delete`
`org delete snapshot`	`force:org:snapshot:delete`
`org display user`	`force:user:display`
`org display`	`force:org:display`
`org generate password`	`force:user:password:generate`
`org get snapshot`	`force:org:snapshot:get`
`org list`	`force:org:list`, `auth:list`
`org list auth`	`auth:list`

`sf`-style Command	Equivalent `sfdx`-style Command
`org list limits`	`force:limits:api:display`
`org list metadata`	`force:mdapi:listmetadata`
`org list metadata-types`	`force:mdapi:describemetadata`
`org list shape`	`force:org:shape:list`
`org list snapshot`	`force:org:snapshot:list`
`org list sobject record-counts`	`force:limits:recordcounts:display`
`org list users`	`force:user:list`
`org login jwt`	`auth:jwt:grant`
`org login web`	`auth:web:login`
`org login access-token`	`auth:accesstoken:store`
`org login device`	`auth:device:login`
`org login sfdx-url`	`auth:sfdxurl:store`
`org logout`	`auth:logout`
`org open`	`force:org:open`
`org resume sandbox`	`force:org:status --wait`
`org resume scratch`	No equivalent.
`package1 version create`	`force:package1:version:create`
`package1 version create get`	`force:package1:version:create:get`
`package1 version display`	`force:package1:version:display`
`package1 version list`	`force:package1:version:list`
`package create`	`force:package:create`
`package delete`	`force:package:delete`
`package install`	`force:package:install`
`package install report`	`force:package:install:report`
`package installed list`	`force:package:installed:list`
`package list`	`force:package:list`
`package uninstall`	`force:package:uninstall`
`package uninstall report`	`force:package:uninstall:report`
`package update`	`force:package:update`
`package version create`	`force:package:version:create`

`sf`-style Command	Equivalent `sfdx`-style Command
`package version create list`	`force:package:version:create:list`
`package version create report`	`force:package:version:create:report`
`package version delete`	`force:package:version:delete`
`package version displayancestry`	`force:package:version:displayancestry`
`package version list`	`force:package:version:list`
`package version promote`	`force:package:version:promote`
`package version report`	`force:package:version:report`
`package version update`	`force:package:version:update`
`project convert mdapi`	`force:mdapi:convert`
`project convert source`	`force:source:convert`
`project delete source`	`force:source:delete`
`project delete tracking`	`force:source:tracking:clear`
`project deploy cancel`	`force:source:deploy:cancel`
`project deploy preview`	`force:source:status`
`project deploy quick`	`force:source:deploy --validateddeployrequestid`
`project deploy report`	`force:source:deploy:report --wait 0`
`project deploy resume`	`force:source:deploy:report`
`project deploy start`	`force:source:deploy`
`project deploy validate`	`force:source:deploy --checkonly --testlevel >NoTestRun`
`project generate`	`force:project:create`
`project generate manifest`	`force:source:manifest:create`
`project list ignored`	`force:source:ignored:list`
`project reset tracking`	`force:source:tracking:reset`
`project retrieve start`	`force:source:retrieve`
`project retrieve preview`	`force:source:status`
`search`	`search`
`schema generate field`	No equivalent.
`schema generate platformevent`	No equivalent.
`schema generate sobject`	No equivalent.

`sf`-style Command	Equivalent `sfdx`-style Command
`schema generate tab`	No equivalent.
`sobject describe`	`force:schema:sobject:describe`
`sobject list`	`force:schema:sobject:list`
`static-resource generate`	`force:staticresource:create`
`version`	`version`
`visualforce generate component`	`force:visualforce:component:create`
`visualforce generate page`	`force:visualforce:page:create`
`which`	`which`

CLI Deprecation Policy

Salesforce deprecates CLI commands and flags when, for example, the underlying API changes.

The Salesforce CLI deprecation policy is:

- Salesforce announces new and upcoming deprecations of commands and flags in the weekly Salesforce CLI release notes.
- Salesforce can deprecate a command or flag at any time.
- When you run the deprecated command, Salesforce provides a deprecation warning for a minimum of 4 months.
- Salesforce removes the deprecated command or flag 4 months, or more, after the deprecation warning first appears.
- If you use a command or flag that's been deprecated but not yet removed, you get a warning message in `stderr` in the human-readable output. If you specify JSON output, the warning is presented as a property. The message includes the plugin version in which we plan to remove the command or flag. The command help also includes deprecation information when appropriate.
- When possible, Salesforce provides a functional alternative to the deprecated command or flag.
- For our policy on changes to a Salesforce CLI command's JSON response, see Support for JSON Responses.

Discover Salesforce Plugins

Check out these other plugins that work with specific Salesforce features. These plugins are created by Salesforce.

ISV Technical Enablement Plugin

The ISVTE plugin is an on-demand Technical Evangelist. It scans your package metadata and code, and provides targeted feedback to help you improve and future-proof your app. The feedback includes a detailed metadata inventory, recommendations on features or technologies to consider using, enablement resources, and installation limitations. The feedback also includes best practices, partner alerts, guidance on improving your partner Trailblazer score, and more. While it's designed for ISV and OEM partners, anyone developing on the platform can use it.

When you install the plugin, you're asked to confirm that it's unsigned. Answer *`yes`*. This behavior is expected.

See GitHub for documentation and more information.

CRM Analytics Plugin

CRM Analytics is a cloud-based platform for connecting data from multiple sources, creating interactive views of that data, and sharing those views in apps.

Use the CRM Analytics CLI plugin to create scratch orgs with Analytics Studio, which you can use to develop and test source code. The plugin includes commands that call a subset of the Analytics REST API endpoints to manage CRM Analytics assets programmatically. Create and iteratively develop CRM Analytics templates. Update and delete apps, dashboards, lenses, and dataflows. Use history commands to restore previous versions of dashboards and dataflows. Manage the auto-install lifecycle for embedded templated apps.

See Develop with the Analytics Plugin for the Salesforce CLI for documentation and more information.

Salesforce Code Analyzer Plugin

The Salesforce Code Analyzer plugin is a unified tool for static analysis of source code, in multiple languages (including Apex), with a consistent command-line interface and report output. We currently support the PMD rule engine, ESLint, and RetireJS.

The plugin creates "rule violations" when the scanner identifies issues. Developers use this information as feedback to fix their code. Integrate this plugin into your continuous integration (CI) solution to continually enforce the rules and ensure high-quality code.

See Salesforce Code Analyzer for documentation and more information.