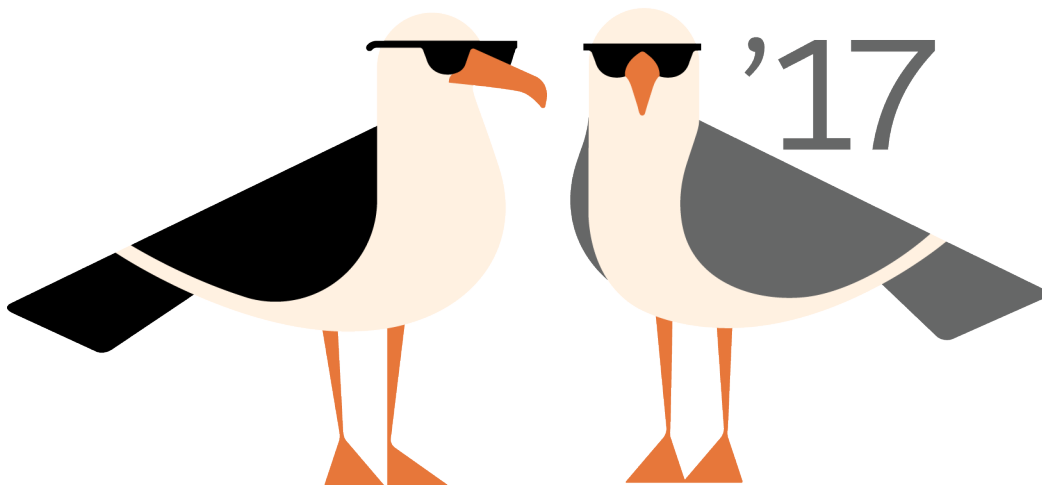




Salesforce DX Developer Guide (Beta)

Version 40.0, Summer '17



CONTENTS

Chapter 1: How Salesforce Developer Experience Changes the Way You Work (Beta)	1
Use a Sample Repo to Get Started	3
Create an Application	3
Migrate or Import Existing Source	4
Chapter 2: Project Setup	5
Salesforce CLI Configuration and Tips	6
CLI Configuration Values	6
CLI Environment Variables	7
Salesforce DX Usernames and Orgs	8
CLI Parameter Resolution Order	11
Support for JSON Responses	11
Log Messages and Log Levels	12
Sample Repository on GitHub	12
Create a Salesforce DX Project	13
Create a Salesforce DX Project from Existing Source	13
Retrieve Source from an Existing Managed Package	14
Retrieve Unpackaged Source Defined in a package.xml File	14
Retrieve Unpackaged Source by Creating a Temporary Unmanaged Package	15
Convert the Metadata API Source	16
Link a Namespace to the Dev Hub Org	16
Salesforce DX Project Configuration	17
Chapter 3: Authorization	19
Authorize an Org Using the Web-Based Flow	20
Authorize an Org Using the JWT-Based Flow	21
Create a Private Key and Self-Signed Digital Certificate	21
Create a Connected App	22
Use an Existing Access Token Instead of Authorizing	23
Authorization Information for an Org	24
Chapter 4: Scratch Orgs	25
Scratch Org Definition File	26
Scratch Org Definition Configuration Values	27
Create Scratch Orgs	30
Supported Metadata API Types	32
Salesforce DX Project Structure and Source File Format	32
Push Source to the Scratch Org	36

Contents

How to Exclude Source When Syncing or Converting	38
Assign a Permission Set	39
Ways to Add Data to Your Scratch Org	39
Example: Export and Import Data Between Orgs	41
Pull Source from the Scratch Org to Your Project	42
Generate a Password for a Scratch Org	43
Manage Scratch Orgs from the Dev Hub	44
Chapter 5: Development	45
Create a Lightning App and Components	47
Create an Apex Class	47
Track Changes Between the Project and Scratch Org	47
Testing	47
Chapter 6: Build and Release Your App	49
Build and Release Your App with Managed Packages	50
Packaging Checklist	50
Deploy the Package Metadata to the Packaging Org	51
Create a Beta Version of Your App	52
Install the Package in a Target Org	53
Create a Managed Package Version of Your App	53
View Information About a Package	54
View All Package Versions in the Org	54
Package IDs	55
Build and Release Your App with Metadata API	55
Deploy Changes to a Sandbox for Validation	56
Release Your App to Production	57
Chapter 7: Continuous Integration	59
Continuous Integration Using Jenkins	60
Configure Your Environment for Jenkins	60
Jenkinsfile Walkthrough	61
Continuous Integration with Travis CI	65
Chapter 8: Troubleshoot Salesforce DX	66
CLI Version Information	67
Run CLI Commands on macOS Sierra (Version 10.12)	67
Error: No defaultdevhubusername org found	67
Unable to Work After Failed Org Authorization	68
Error: Lightning Experience-Enabled Custom Domain Is Unavailable	68
Chapter 9: Considerations for Salesforce DX (Beta)	70

CHAPTER 1 How Salesforce Developer Experience Changes the Way You Work (Beta)

In this chapter ...

- [Use a Sample Repo to Get Started](#)
- [Create an Application](#)
- [Migrate or Import Existing Source](#)

Salesforce Developer Experience (DX) is a new way to manage and develop apps on the Force.com platform across their entire life cycle. It brings together the best of Force.com to enable source-driven development, team collaboration with governance, and new levels of agility for custom app development on Salesforce.



Note: This release contains a beta version of Salesforce DX, which means it's a high-quality feature with known limitations. Salesforce DX isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only on the basis of generally available products and features. You can provide feedback and suggestions for Salesforce DX in the [Salesforce DX Beta](#) group in the Success Community.

Highlights of Salesforce DX include:

- Your tools, your way. With Salesforce DX, you use the developer tools you already know.
- The ability to apply best practices to software development. Source code and metadata exist outside of the org and provide more agility to develop Salesforce apps in a team environment. Instead of the org, your version control system is the source of truth.
- A powerful command-line interface (CLI) removes the complexity of working with your Salesforce org for development, continuous integration, and delivery.
- Flexible and configurable scratch orgs that you build for development and automated environments. This new type of org makes it easier to build your apps and packages.
- You can use any IDE or text editor you want with the CLI and externalized source.
- If you are using Eclipse, an updated Eclipse IDE plug-in built specifically for Salesforce DX accelerates app development.

Are You Ready to Begin?

Here's the basic order for doing your work using Salesforce DX. These workflows include the most common CLI commands. For all commands, see the *Salesforce CLI Command Reference (Beta)*.

- [Use a Sample Repo to Get Started](#) on page 3
- [Create an Application](#) on page 3

How Salesforce Developer Experience Changes the Way You Work (Beta)

- [Migrate or Import Existing Source](#) on page 4

SEE ALSO:

[Salesforce DX \(Salesforce Developer Center Web Site\)](#)

[Salesforce DX = UX for Developers \(Salesforce Developer Blog\)](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Use a Sample Repo to Get Started

The quickest way to get going with Salesforce DX is to clone the `sfdx-simple` GitHub repo. Use its configuration files and Force.com application to try some commonly used Salesforce CLI commands.

1. Open a terminal or command prompt window, and clone the `sfdx-simple` GitHub sample repo using HTTPS or SSH.

```
git clone https://github.com/forcedotcom/sfdx-simple.git
--or--
git clone git@github.com:forcedotcom/sfdx-simple.git
```

2. Change to the `sfdx-simple` project directory.

```
cd sfdx-simple
```

3. Authorize your Developer Hub (Dev Hub) org, set it as your default, and assign it an alias.

```
sfdx force:auth:web:login --setdefaultdevhubusername --setalias DevHub
```

Enter your Dev Hub org credentials in the browser that opens. After you log in successfully, you can close the browser.

4. Create a scratch org using the `config/project-scratch-def.json` file, set the username as your default, and assign it an alias.

```
sfdx force:org:create --setdefaultusername -f config/project-scratch-def.json --setalias my-scratch-org
```

5. Push source and tests, located in the `force-app` directory, to the scratch org.

```
sfdx force:source:push
```

6. Run Apex tests.

```
sfdx force:apex:test:run --resultformat human
```

7. Open the scratch org and view the pushed metadata under Most Recently Used.

```
sfdx force:org:open
```

SEE ALSO:

[Sample Repository on GitHub](#)

[Authorization](#)

[Create Scratch Orgs](#)

[Push Source to the Scratch Org](#)

[Testing](#)

Create an Application

Follow the basic workflow when you are starting from scratch to create and develop an app that runs on the Force.com platform.

1. [Set up your project.](#) on page 5


2. [Authorize the Developer Hub org for the project.](#) on page 19
3. [Configure your local project.](#) on page 17
4. [Create a scratch org.](#) on page 30
5. [Push the source from your project to the scratch org.](#) on page 36
6. [Develop the app.](#) on page 45
7. [Pull the source to keep your project and scratch org in sync.](#) on page 42
8. [Run tests.](#) on page 47
9. Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- [Build and release your app with managed packages](#) on page 50
- [Build and release your app using the Metadata API](#) on page 55

Migrate or Import Existing Source

Use the Metadata API to retrieve the code, and then convert your source for use in a Salesforce DX project.

 **Tip:** If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip the retrieve step and go directly to converting the source.

1. [Set up your project.](#) on page 5
2. [Retrieve your metadata.](#) on page 13
3. [Convert the MDAPI source you just retrieved to Salesforce DX project format.](#) on page 16
4. [Authorize the Developer Hub org for the project.](#) on page 19
5. [Configure your local project.](#) on page 17
6. [Create a scratch org.](#) on page 30
7. [Push the source from your project to the scratch org.](#) on page 36
8. [Develop the app.](#) on page 45
9. [Pull the source to sync your project and scratch org.](#) on page 42
10. [Run tests.](#) on page 47
11. Add, commit, and push changes. Create a pull request.

Deploy your app using one of the following methods:

- [Build and release your app with managed packages.](#) on page 50
- [Build and release your app using the Metadata API.](#) on page 55

CHAPTER 2 Project Setup

In this chapter ...

- [Salesforce CLI Configuration and Tips](#)
- [Sample Repository on GitHub](#)
- [Create a Salesforce DX Project](#)
- [Create a Salesforce DX Project from Existing Source](#)
- [Retrieve Source from an Existing Managed Package](#)
- [Retrieve Unpackaged Source Defined in a package.xml File](#)
- [Retrieve Unpackaged Source by Creating a Temporary Unmanaged Package](#)
- [Convert the Metadata API Source](#)
- [Link a Namespace to the Dev Hub Org](#)
- [Salesforce DX Project Configuration](#)

Salesforce DX introduces a new project structure for your org's metadata (code and configuration), your org templates, your sample data, and all your team's tests. Store these items in a version control system (VCS) to bring consistency to your team's development processes. Retrieve the contents of your team's repository when you're ready to develop a new feature.

You can use your preferred VCS. Most of our examples use Git.

You have different options to create a Salesforce DX project depending on how you want to begin.

Use the Sample Repository on GitHub on page 12	Explore the features of Salesforce DX using one of our sample repos and your own VCS and toolset.
Create a Salesforce DX Project from Existing Source on page 13	Start with an existing Salesforce app to create a Salesforce DX project.
Create a Salesforce DX Project on page 13	Create an app on the Force.com platform using a Salesforce DX project.

Salesforce CLI Configuration and Tips

Use the Salesforce command-line interface (CLI) for most Salesforce DX tasks. These tasks include authorizing a Dev Hub org, creating a scratch org, synchronizing source code between your scratch orgs and VCS, and running tests.

You can start using the CLI right after you install it.

The CLI commands are grouped into top-level topics. For example, the `force` top-level topic is divided into topics that group commands by functionality, such as the `force:org` commands to manage your orgs.

Run `--help` at each level to get more information.

```
sfdx --help // lists all top-level topics
sfdx force --help // lists all the topics under force
sfdx force:org --help // lists all the commands in the topic force:org
sfdx force:org:open --help // detailed info about the force:org:open command
```

Run this command to view all available commands in the `force` topic.

```
sfdx force:doc:commands:list
```

[CLI Configuration Values](#)

You can set CLI configuration values for your current project or for all projects.

[CLI Environment Variables](#)

You can set environment variables to configure some CLI behaviors.

[Salesforce DX Usernames and Orgs](#)

Many CLI commands connect to an org to complete their task. For example, the `force:org:create` command, which creates a scratch org, connects to a Dev Hub org. The `force:source:push|pull` commands synchronize source code between your project and a scratch org. In each case, the CLI command requires a username to determine which org to connect to. Usernames are unique within the entire Salesforce ecosystem and have a one-to-one association with a specific org.

[CLI Parameter Resolution Order](#)

Because you can specify parameters for a given CLI command in several ways, it's important to know the order of parameter resolution.

[Support for JSON Responses](#)

Salesforce CLI commands typically display their output to the console (stdout) in non-structured, human-readable format. Messages written to the log file (stderr) are always in JSON format.

[Log Messages and Log Levels](#)

The Salesforce CLI writes all log messages to the `USER_HOME_DIR/.sfdx/sfdx.log` file. CLI invocations append log messages to this running log file. Only errors are output to the terminal or command window from which you run the CLI.

SEE ALSO:

[Salesforce DX Setup Guide \(Beta\)](#)

CLI Configuration Values

You can set CLI configuration values for your current project or for all projects.

To set a configuration value for the current project:

```
sfdx force:config:set name=<value>
```

To set the value for all your projects:

```
sfdx force:config:set name=<value> --global
```

You can view the local and global configuration values that you have set. The output lists the local values for the project directory from which you are running the command and all global values.

```
sfdx force:config:list
```

To return one or more previously set configuration values, use `force:config:get`. It is often useful to specify JSON output for this command for easier parsing in a continuous integration (CI) environment. For example, to return the value of `defaultusername` and `defaultdevhubusername`:


```
sfdx force:config:get defaultusername defaultdevhubusername --json
```

To unset a configuration value, set it to no value. For example, to unset the `instanceUrl` configuration value:

```
sfdx force:config:set instanceUrl=
```

You can set these CLI configuration values.

Configuration Value Name	Description
<code>defaultusername</code>	The username for an org that all commands run against by default.
<code>defaultdevhubusername</code>	The username of your Dev Hub org that the <code>force:org:create</code> command defaults to.
<code>instanceUrl</code>	The URL of the Salesforce instance that is hosting your org.

 **Warning:** The Salesforce CLI stores local configuration values in `PROJECT_DIR/.sfdx/sfdx-config.json` and global values in `USER_HOME_DIR/.sfdx/sfdx-config.json`. Do not edit or remove these files. Use `force:config:list` to view CLI configuration information.

SEE ALSO:

[Salesforce DX Usernames and Orgs](#)

[Authorization](#)

[Use an Existing Access Token Instead of Authorizing](#)

CLI Environment Variables

You can set environment variables to configure some CLI behaviors.

Environment Variable	Description
<code>SFDX_LOG_LEVEL</code>	Sets the level of messages that the CLI writes to the log file.
<code>SFDX_CONTENT_TYPE</code>	All CLI commands output results in JSON format.
<code>SFDX_USE_GENERIC_UNIX_KEYCHAIN</code>	(Linux and macOS only) Set to <code>true</code> if you want to use the generic Unix keychain instead of the Linux <code>libsecret</code> library or macOS

Environment Variable	Description
	keychain. Specify this variable when using the CLI with ssh or "headless" in a CI environment.
SFDX_AUTOUPDATE_DISABLE	Set to <code>true</code> to disable the auto-update feature of the CLI. By default the CLI periodically checks for and installs updates.
SFDX_DOMAIN_RETRY	Specifies the time, in seconds, that the CLI waits for the Lightning Experience custom domain to resolve and become available in a newly-created scratch org. The default value is 240 (4 minutes). Set the variable to 0 to bypass the Lightning Experience custom domain check entirely.

SEE ALSO:

[Log Messages and Log Levels](#)

[Support for JSON Responses](#)

Salesforce DX Usernames and Orgs

Many CLI commands connect to an org to complete their task. For example, the `force:org:create` command, which creates a scratch org, connects to a Dev Hub org. The `force:source:push|pull` commands synchronize source code between your project and a scratch org. In each case, the CLI command requires a username to determine which org to connect to. Usernames are unique within the entire Salesforce ecosystem and have a one-to-one association with a specific org.



Note: The examples in this topic might refer to CLI commands that you are not yet familiar with. For now, focus on how to specify the usernames, configure default usernames, and use aliases. The CLI commands are described later.

When you create a scratch org, the CLI generates a username. The username looks like an email address, such as `test-gjt2ycpivtpz@your_company.net`. You do not need a password to connect to or open a scratch org, although you can generate one later with the `force:user:password:generate` command.

Salesforce recommends that you set a default username for the orgs that you connect to the most during development. The easiest way to do this is when you authorize a Dev Hub org or create a scratch org. Specify the `--setDefaultDevHubUsername` or `--setDefaultUsername` parameter, respectively, from within a project directory. You can also create an alias to give the usernames more readable names. You can use usernames or their aliases interchangeably for all CLI commands that connect to an org.

These examples set the default usernames and aliases when you authorize an org and then when you create a scratch org.

```
sfdx force:auth:web:login --setDefaultDevHubUsername --setAlias my-hub-org
sfdx force:org:create --definitionfile my-org-def.json --setDefaultUsername --setAlias my-scratch-org
```

To verify whether a CLI command requires an org connection, look at its parameter list with the `--help` parameter. Commands that have the `--targetDevHubUsername` parameter connect to the Dev Hub org. Similarly, commands that have `--targetUsername` connect to scratch orgs, sandboxes, and so on. This example displays the parameter list and help information about `force:org:create`.

```
sfdx force:org:create --help
```

When you run a CLI command that requires an org connection and you don't specify a username, the command uses the default. To see your default usernames, run `force:org:list` to display all the orgs you've authorized or created. The default Dev Hub and scratch orgs are marked on the left with (D) and (U), respectively.

Let's run through a few examples to see how this works. This example pushes source code to the scratch org that you've set as the default.

```
sfdx force:source:push
```

To specify an org other than the default, use `--targetusername`. For example, let's say you created another scratch org with alias `my-other-scratch-org`. It's not the default but you still want to push source to it.

```
sfdx force:source:push --targetusername my-other-scratch-org
```

This example shows how to use the `--targetdevhubusername` parameter to specify a non-default Dev Hub org when creating a scratch org.

```
sfdx force:org:create --targetdevhubusername jdoe@mydevhub.com --definitionfile my-org-def.json --setalias yet-another-scratch-org
```

More About Setting Default Usernames

If you've already created a scratch org, you can set the default username with the `force:config:set` command from your project directory.

```
sfdx force:config:set defaultusername=test-ymlqf29req5@your_company.net
```

The command sets the value locally, so it works only for the current project. To use the default username for all projects on your computer, specify the `--global` parameter. You can run this command from any directory. Local project defaults override global defaults.

```
sfdx force:config:set defaultusername=test-ymlqf29req5@your_company.net --global
```

The process is similar to set a default Dev Hub org, except you use the `defaultdevhubusername` config value.

```
sfdx force:config:set defaultdevhubusername=jdoe@mydevhub.com
```

More About Aliasing

Use the `force:alias:set` command to set an alias for an org or after you've authorized an org. You can create an alias for any org: Dev Hub, scratch, production, sandbox, and so on. So when you issue a command that requires the org username, using an alias for the org that you can easily remember can speed up things.

```
sfdx force:alias:set my-scratch-org=test-ymlqf29req5@your_company.net
```

An alias also makes it easy to set a default username. The previous example of using `force:config:set` to set `defaultusername` now becomes much more digestible when you use an alias rather than the username.

```
sfdx force:config:set defaultusername=my-scratch-org
```

Set multiple aliases with a single command by separating the name-value pairs with a space.

```
sfdx force:alias:set org1=<username> org2=<username>
```

You can associate an alias with only one username at a time. If you set it multiple times, the alias points to the most recent username. For example, if you run the following two commands, the alias `my-org` is set to `test-ymmlqf29req5@your_company.net`.

```
sfdx force:alias:set my-org=test-blahdiblah@whoanellie.net
sfdx force:alias:set my-org=test-ymmlqf29req5@your_company.net
```

To view all aliases that you've set, use one of the following commands.

```
sfdx force:alias:list
sfdx force:org:list
```

To remove an alias, set it to nothing.

```
sfdx force:alias:set my-org=
```

List All Your Orgs

Use the `force:org:list` command to display the usernames for the orgs that you've authorized and the active scratch orgs that you've created.

```
sfdx force:org:list
=== Orgs
  ALIAS          USERNAME          ORG ID          CONNECTED STATUS
  -----
  DD-ORG        jdoe@dd-204.com   00D...OEA       Connected
  (D) devhuborg208 jdoe@mydevhub.com 00D...MAC       Connected

  ALIAS          SCRATCH ORG NAME USERNAME          ORG ID          EXPIRATION DATE
  -----
  my-scratch Your Company      test-ymmlqf5@a_company.net 00D...UAI       2017-06-13
  (U) scratch208 Your Company      test-gjt2ycz@b_company.net 00D...UAY       2017-06-13
```

The top section of the output lists the non-scratch orgs that you've authorized, including Dev Hub orgs, production orgs, and sandboxes. The output displays the usernames that you specified when you authorized the orgs, their aliases, their IDs, and whether the CLI can connect to it. A (D) on the left points to the default Dev Hub username.

The lower section lists the active scratch orgs that you've created and their usernames, org IDs, and expiration dates. A (U) on the left points to the default scratch org username.

To view more information about scratch orgs, such as the create date, instance URL, and associated Dev Hub org, use the `--verbose` parameter.

```
sfdx force:org:list --verbose
```

Use the `--clean` parameter to remove non-active scratch orgs from the list. The command prompts you before it does anything.

```
sfdx force:org:list --clean
```

SEE ALSO:

[Authorization](#)

[Scratch Org Definition File](#)

[Create Scratch Orgs](#)

[Generate a Password for a Scratch Org](#)

[Push Source to the Scratch Org](#)

CLI Parameter Resolution Order

Because you can specify parameters for a given CLI command in several ways, it's important to know the order of parameter resolution.

The order of precedence for parameter resolution is:

1. Command-line parameters, such as `--loglevel`, `--targetusername`, or `--targetdevhubusername`.
2. Parameters listed in a file specified by the command line. An example is a scratch org definition in a file specified by the `--definitionfile` parameter of `force:org:create`.
3. Environment variables, such as `SFDX_LOG_LEVEL`.
4. Local CLI configuration values, such as `defaultusername` or `defaultdevhubusername`. To view the local values, run `force:config:list` from your project directory.
5. Global CLI configuration values. To view the global values, run `force:config:list` from any directory.

For example, if you set the `SFDX_LOG_LEVEL` environment variable to `INFO` but specify `--loglevel DEBUG` for a command, the log level is `DEBUG`. This behavior happens because command-line parameters are at the top of the precedence list.

If you specify the `--targetusername` parameter for a specific CLI command, the CLI command connects to an org with that username. It does not connect to an org using the `defaultusername`, assuming that you set it previously with the `force:config:set` command.

Support for JSON Responses

Salesforce CLI commands typically display their output to the console (`stdout`) in non-structured, human-readable format. Messages written to the log file (`stderr`) are always in JSON format.

To view the console output in JSON format, specify the `--json` parameter for a particular CLI command.

```
sfdx force:org:display --json
```

Most CLI commands support JSON output. To confirm, run the command with the `--help` parameter to view the supported parameters.

To get JSON responses to all Salesforce CLI commands without specifying the `--json` option each time, set the `SFDX_CONTENT_TYPE` environment variable.

```
export SFDX_CONTENT_TYPE=JSON
```

Log Messages and Log Levels

The Salesforce CLI writes all log messages to the `USER_HOME_DIR/.sfdx/sfdx.log` file. CLI invocations append log messages to this running log file. Only errors are output to the terminal or command window from which you run the CLI.

The default level of log messages is ERROR. You can set the log level to one of the following, listed in order of least to most information. The level is cumulative: for the DEBUG level, the log file also includes messages at the INFO, WARN, and ERROR levels.

- ERROR
- WARN
- INFO
- DEBUG
- TRACE


You can change the log level in two ways, depending on what you want to accomplish.

To change the log level for the execution of a single CLI command, use the `--loglevel` parameter. Changing the log level in this way does not affect subsequent CLI use. This example specifies debug-level log messages when you create a scratch org.

```
sfdx force:org:create --definitionfile config/project-scratch-def.json --loglevel DEBUG --setalias my-scratch-org
```

To globally set the log level for all CLI commands, set the `SFDX_LOG_LEVEL` environment variable. For example, on UNIX:

```
export SFDX_LOG_LEVEL=DEBUG
```

 **Note:** The Salesforce CLI gathers diagnostic information about its use and reports it to Salesforce so that the development team can investigate issues. The type of information includes command duration and command invocation counts.

Sample Repository on GitHub

If you want to check out Salesforce DX features quickly, start with the `sfdx-simple` GitHub repo. It contains an example of the project configuration file (`sfdx-project.json`), a simple Force.com app, and Apex tests.

Cloning this repo creates the directory `sfdx-simple`. See the repo's Readme for more information.

Assuming that you've already set up Git, use the `git clone` command to clone the master branch of the repo from the command line.

To use HTTPS:

```
git clone https://github.com/forcedotcom/sfdx-simple.git
```

To use SSH:

```
git clone git@github.com:forcedotcom/sfdx-simple.git
```

If you don't want to use Git, download a .zip file of the repository's source using Clone, or download on the GitHub website. Unpack the source anywhere on your local file system.

To check out a more complex example, clone the `sfdx-dreamhouse` GitHub repo. This standalone application contains multiple Apex classes, Lightning components, Visualforce components, and custom objects.

SEE ALSO:

[sfdx-simple Sample GitHub Repo](#)

[sfdx-dreamhouse Sample GitHub Repo](#)

Create a Salesforce DX Project

A Salesforce DX project has a specific structure and a configuration file that identifies the directory as a Salesforce DX project.

1. Use the `force:project:create` command to create a skeleton project structure for your Salesforce DX project. If you don't indicate an output directory, the project directory is created in the current location. You can also specify the default package directory to target when syncing source to and from the scratch org. If you don't indicate a default package directory, this command creates a default package directory, `force-app`.

Example:

```
sfdx force:project:create --projectname mywork
sfdx force:project:create --projectname mywork --defaultpackagedir myapp
```

- (Optional) Register the namespace with the Dev Hub org.
- Configure the project (`sfdx-project.json`). If you use a namespace, update this file to include it.
- Create a scratch org definition that produces scratch orgs that mimic the shape of another org you use in development, such as sandbox, packaging, or production. The `config` directory of your new project contains a sample scratch org definition file.

SEE ALSO:

[Create a Salesforce DX Project from Existing Source](#)


[Salesforce DX Project Configuration](#)

[Link a Namespace to the Dev Hub Org](#)

[Scratch Org Definition File](#)

Create a Salesforce DX Project from Existing Source

If you are already a Salesforce developer or ISV, you likely have existing source in a managed package in your packaging org or some application source in your sandbox or production org. Before you begin using Salesforce DX, retrieve the existing source and convert it to the Salesforce DX project format.

 **Tip:** If your current repo follows the directory structure that is created from a Metadata API retrieve, you can skip to converting the Metadata API source after you create a Salesforce DX project.

1. Create a Salesforce DX project.
2. Create a directory for the metadata retrieve. You can create this directory anywhere.

```
mkdir mdapipkg
```

3. Retrieve your metadata source.

Format of Current Source	How to Retrieve Your Source for Conversion
You are a partner who has your source already defined as a managed package in your packaging org.	Retrieve Source from an Existing Managed Package on page 14
You have a <code>package.xml</code> file that defines your unpackaged source.	Retrieve Unpackaged Source Defined in a package.xml File on page 14
You don't have your source defined in a package.	Retrieve Unpackaged Source by Creating a Temporary Unmanaged Package on page 15

SEE ALSO:

[Convert the Metadata API Source](#)

[Create a Salesforce DX Project](#)

Retrieve Source from an Existing Managed Package

If you're a partner or ISV who already has a managed package in a packaging org, you're in the right place. You can retrieve that package, unzip it to your local project, and then convert it to Salesforce DX format, all from the CLI.

Before you begin, create a Salesforce DX project.

1. In the project, create a folder to store what's retrieved from your org, for example, `mdapipkg`.
2. Retrieve the metadata.

```
sfdx force:mdapi:retrieve -s -r ./mdapipkg -u <username> -p <package name>
```

The username can be a username or alias for the target org (such as a packaging org) from which you're pulling metadata. The `-s` parameter indicates that you're retrieving a single package.

3. Check the status of the retrieve.

```
sfdx force:mdapi:retrieve -u <username> -i jobid
```

4. Unzip the zip file.
5. (Optional) Delete the zip file.

After you finish, convert the metadata source.

SEE ALSO:

[Create a Salesforce DX Project](#)

[Convert the Metadata API Source](#)

Retrieve Unpackaged Source Defined in a package.xml File

If you already have a `package.xml` file, you can retrieve it, unzip it in your local project, and convert the source to Salesforce DX format. You can do all these tasks from the CLI. The `package.xml` file defines the source you want to retrieve.

If you already have the source retrieved from the Metadata API, you can skip to converting the metadata API source.

1. In the project, create a folder to store what's retrieved from your org, for example, `mdapipkg`.
2. Retrieve the metadata.

```
sfdx force:mdapi:retrieve -r ./mdapipkg -u <username> -k ./package.xml
```

The username can be the scratch org username or an alias. The `-k` parameter indicates the path to the `package.xml` file, which is the unpackaged manifest of components to retrieve.

3. Check the status of the retrieve.

```
sfdx force:mdapi:retrieve -u <username> -i jobid
```

4. Unzip the zip file.
5. (Optional) Delete the zip file.

After you retrieve the source and unzip it, you no longer need the zip file, so you can delete it.

After you finish, convert the metadata source.

SEE ALSO:

[Convert the Metadata API Source](#)

Retrieve Unpackaged Source by Creating a Temporary Unmanaged Package

Don't already have your source defined in a `package.xml` file? No worries. To simplify the source retrieval process, you can use the Unmanaged Package UI to define what you want to retrieve. Create an unmanaged package in your sandbox, and add your app's source to it.

1. Open the sandbox org that contains your metadata.

```
sfdx force:org:open -u <username> --path one/one.app
```

You can use the sandbox username or an alias.

2. Open **All Setup**.
3. Open **Package Manager**.
4. Create a package with the name of your application.
5. Add the metadata source to the package.
6. In your Salesforce DX project, create a folder for the MDAPI retrieve, for example, `mdapipkg`.
7. Retrieve the metadata.

```
sfdx force:mdapi:retrieve -s -r ./mdapipkg -u <username> -p <package name>
```

The `-s` parameter indicates that you're retrieving a single package. The package name is the name you used in step 4.

8. Check the status of the retrieve.

```
sfdx force:mdapi:retrieve -u <username> -r ./mdapipkg -i jobid
```

9. Unzip the zip file.

10. (Optional) Delete the zip file.

After you retrieve your source, you can delete or refresh your sandbox or delete the package definition in the sandbox org. Now that you have the source, you don't need the resources.

Next, convert the metadata source.

SEE ALSO:

[Convert the Metadata API Source](#)

Convert the Metadata API Source

After you retrieve the source from your org, you can complete the configuration of your project and convert the source to Salesforce DX project format.

The convert command ignores all files that start with a "dot," such as `.DS_Store`. To exclude more files from the convert process, add a `.forceignore` file.

1. To indicate which package directory is the default, update the `sfdx-project.json` file.
2. Convert the metadata API source to Salesforce DX project format.

```
sfdx force:mdapi:convert --rootdir <retrieve dir name>
```

The `--rootdir` parameter is the name of the directory that contains the metadata source, that is, one of the package directories or subdirectories defined in the `sfdx-project.json` file.

If you don't indicate an output directory with the `--outputdir` parameter, the converted source is stored in the default package directory indicated in the `sfdx-project.json` file. If the output directory is located outside of the project, you can indicate its location using an absolute path.

Next steps:

- Authorize the Dev Hub org and set it as the default
- Configure the Salesforce DX project
- Create a scratch org

SEE ALSO:

[How to Exclude Source When Syncing or Converting](#)

[Salesforce DX Project Configuration](#)

[Authorization](#)

[Create Scratch Orgs](#)

Link a Namespace to the Dev Hub Org

To use a namespace with a scratch org, you must link the Developer Edition org where the namespace is registered to the Dev Hub.

Complete these tasks before you link a namespace.

- If you don't have an org with a registered namespace, create a Developer Edition org that is separate from the Dev Hub or scratch orgs. If you already have an org with a registered namespace, go to Step 1.
- In the Developer Edition org, create and register the namespace.

! **Important:** If you're trying out this feature, choose a disposable namespace. Don't choose a namespace that you want to use in the future for a production org. If you're registering a namespace for a production org, choose it carefully. Once you associate a namespace with an org, you can't change it or reuse it.

1. Log in to your Dev Hub org as the System Administrator or as a user with the Salesforce DX permissions.
2. If you have not already done so, define and deploy a My Domain subdomain.
3. If you are using Salesforce Classic, switch to Lightning Experience. (Linking a namespace works only in Lightning Experience.)
4. From the App Launcher menu, select **Namespace Registry**.
5. Click **Link Namespace**.
6. Log in to the Developer Edition org in which your namespace is registered using the org's System Administrator's credentials.
7. To see the namespace that you linked, create a view on the **Namespace Registry** tab that displays all the linked namespaces.

By default, the **Namespace Registry** tab displays only the recently viewed linked namespaces so your new linked namespace does not automatically display.

SEE ALSO:

[Create a Developer Edition Org](#)

[Create a Namespace in Your Org](#)

[Add Salesforce DX Users \(Salesforce DX Setup Guide \(Beta\)\)](#)

[Define Your Domain Name](#)

[Test and Deploy Your New My Domain Subdomain](#)

Salesforce DX Project Configuration

The project configuration file `sfdx-project.json` indicates that the directory is a Salesforce DX project. The configuration file contains project information and facilitates the authentication of scratch orgs. It also tells Salesforce DX where to put files when syncing between the project and scratch org.

We provide sample `sfdx-project.json` files in the sample repos for creating a project using the CLI or IDE.

We recommend that you check in this file with your source.

```
{
  "packageDirectories" : [
    { "path": "force-app", "default": true},
    { "path" : "unpackaged" },
    { "path" : "utils" }
  ],
  "namespace": "",
  "sfdcLoginUrl" : "https://login.salesforce.com",
  "sourceApiVersion": "40.0"
}
```

You can manually edit these parameters.

packageDirectories

Package directories indicate which directories to target when syncing source to and from the scratch org. These directories can contain source from your managed package, unmanaged package, or unpackaged source, for example, ant tool or change set.

Keep these things in mind when working with package directories.


- The location of the package directory is relative to the project. Don't specify an absolute path. The following two examples are equivalent.

```
"path": "helloWorld"
"path" : "./helloWorld"
```

- You can have only one default path (package directory). If you have only one path, we assume it's the default, so you don't need to explicitly set the `Default` parameter. If you have multiple paths, you must indicate which one is the default.
- The CLI uses the default package directory as the target directory when pulling changes in the scratch org to sync the local project.
- If you do not specify an output directory, the default package directory is also where files are stored during source conversions. Source conversions are both from Metadata API format to Salesforce DX project format and from project format to Metadata API format.

namespace

The global namespace that is used with a package. The namespace must be registered with an org that is associated with your Dev Hub org. This namespace is assigned to scratch orgs created with the `org:create` command.

 **Important:** Register the namespace with Salesforce and then connect the org with the registered namespace to the Dev Hub org.

sfdcLoginUrl

The login URL that the `force:auth` commands use. The default is `login.salesforce.com`. Override the default value if you want users to authorize to a specific Salesforce instance. For example, if you want to authorize into a sandbox org, set this parameter to `test.salesforce.com`.

If you do not specify a default login URL here, or if you run `force:auth` outside the project, you specify the instance URL when authorizing the org.

sourceApiVersion

The API version that the source is compatible with. The `sourceApiVersion` determines the fields retrieved for each metadata type. For example, an icon field was added to the CustomTab for API version 14.0. If you retrieve components for version 13.0 or earlier, the components do not include the icon field.

SEE ALSO:

[Link a Namespace to the Dev Hub Org](#)

[Authorization](#)

[How to Exclude Source When Syncing or Converting](#)

[Pull Source from the Scratch Org to Your Project](#)


[Push Source to the Scratch Org](#)

CHAPTER 3 Authorization

In this chapter ...

- [Authorize an Org Using the Web-Based Flow](#)
- [Authorize an Org Using the JWT-Based Flow](#)
- [Create a Private Key and Self-Signed Digital Certificate](#)
- [Create a Connected App](#)
- [Use an Existing Access Token Instead of Authorizing](#)
- [Authorization Information for an Org](#)

The Dev Hub allows you to create, delete, and manage your Salesforce scratch orgs. After you set up your project on your local machine, you authorize with the Dev Hub org before you can create a scratch org.


 **Note:** The supported editions for Dev Hub orgs are Enterprise Edition (EE) and Unlimited Edition (UE).

You can also authorize other existing orgs, such as sandbox or packaging orgs, to provide more flexibility when using CLI commands. For example, after developing and testing an application using scratch orgs, you can deploy the changes to a centralized sandbox. Or, you can export a subset of data from an existing production org and import it into a scratch org for testing purposes.

You authorize an org only once. To switch between orgs during development, specify your username for the org. Use either the `--targetusername` (or `--targetdevhubusername`) CLI command parameter, set a default username, or use an alias.

You have some options when configuring authentication depending on what you're trying to accomplish.

- We provide the OAuth Refresh Token flow, also called web-based flow, through a global out-of-the-box connected app. When you authorize an org from the command line, you enter your credentials and authorize the global connected app through the Salesforce web browser authentication flow.
- For continuous integration or automated environments in which you don't want to manually enter credentials, use the OAuth JSON Web Tokens (JWT) Bearer Token flow, also called JWT-based flow. This authentication flow is ideal for scenarios where you cannot interactively log in to a browser, such as a continuous integration script.

 **Important:** Credentials are encrypted and stored in the `USER_HOME_DIR/.sfdx` directory, along with other authentication files. These files are used internally by the Salesforce CLI. Do not remove or edit them.

SEE ALSO:

- [Authorize an Org Using the Web-Based Flow](#)
- [Authorize an Org Using the JWT-Based Flow](#)
- [Salesforce DX Usernames and Orgs](#)

Authorize an Org Using the Web-Based Flow

To authorize an org with the web-based flow, all you do is run a CLI command. Enter your credentials in a browser, and you're up and running!

Authorization requires a connected app. We provide a connected app that is used by default. But you can optionally create a connected app if you need more security or control, such as setting the refresh token timeout or specifying IP ranges.

1. (Optional) Create a connected app if you require more security and control than offered by the provided connected app. Enable OAuth settings for the new connected app. Make note of the consumer key because you need it later.
2. If the org you are authorizing is on a My Domain subdomain, update your project configuration file (`sfdx-project.json`). Set the `sfdcLoginUrl` parameter to your My Domain login URL. If you are authorizing a sandbox, set the parameter to `https://test.salesforce.com`. For example:

```
"sfdcLoginUrl" : "https://test.salesforce.com"
```

or

```
"sfdcLoginUrl" : "https://somethingcool.my.salesforce.com"
```

Alternatively, use the `--instanceUrl` parameter of the `force:auth:web:login` command, as shown in the next step, to specify the URL.

3. Run the `force:auth:web:login` CLI command. If you are authorizing a Dev Hub, use the `--setDefaultDevHubUsername` parameter if you want the Dev Hub to be the default for commands that accept the `--targetDevHubUsername` parameter.

```
sfdx force:auth:web:login --setDefaultDevHubUsername --setAlias my-hub-org
sfdx force:auth:web:login --setAlias my-sandbox
```

If you are using your own connected app, use the `--clientId` parameter. For example, if your client identifier (also called the consumer key) is `04580y4051234051` and you are authorizing a Dev Hub:

```
sfdx force:auth:web:login --clientId 04580y4051234051 --setDefaultDevHubUsername
--setAlias my-hub-org
```

To specify a login URL other than the default, such as `https://test.salesforce.com`:

```
sfdx force:auth:web:login --setAlias my-hub-org --instanceUrl https://test.salesforce.com
```

⚠ Important: Use the `--setDefaultDevHubUsername` parameter only when authorizing to a Dev Hub. Do not use it when authorizing to other orgs, such as a sandbox.

4. In the browser window that opens, sign in to your org with your credentials.
5. Close the browser window, unless you want to explore the org.

SEE ALSO:

[Create a Connected App](#)

[Salesforce DX Project Configuration](#)

Authorize an Org Using the JWT-Based Flow

Continuous integration (CI) environments are fully automated and don't support the human interactivity of a web-based flow. In this case, you must use the JWT-based flow to authorize a Dev Hub.

The JWT-based authorization flow requires first generating a digital certificate and creating a connected app. You execute these tasks only once. After that, you can authorize the Dev Hub in a script that runs in your CI environment.

For information about using JWT-based authorization with the Travis CI, see the *Continuous Integration Using Salesforce DX* Trailhead module.

1. If you do not have your own private key and digital certificate, use [OpenSSL to create the key and a self-signed certificate](#). It is assumed in this task that your private key file is named `server.key` and your digital certificate is named `server.crt`.
2. [Create a connected app, and configure it for Salesforce DX](#). This task includes uploading the `server.crt` digital certificate file. Make note of the consumer key when you save the connected app because you need it later.
3. If the org you are authorizing is not hosted on `https://login.salesforce.com`, update your project configuration file (`sfdx-project.json`). Set the `sfdcLoginUrl` parameter to the login URL. Examples of other login URLs are your custom subdomain or `https://test.salesforce.com` for sandboxes. For example:

```
"sfdcLoginUrl" : "https://test.salesforce.com"
```

Alternatively, use the `--instanceUrl` parameter of the `force:auth:jwt:grant` command, as shown in the next step, to specify the URL.

4. Run the `force:auth:jwt:grant` CLI command. Specify the client identifier from your connected app (also called the consumer key), the path to the private key file (`server.key`), and the JWT authentication username. When you authorize a Dev Hub, set it as the default with the `--setDefaultDevHubUsername` parameter. For example:

```
sfdx force:auth:jwt:grant --clientid 04580y4051234051 --jwtkeyfile
/Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org --setDefaultDevHubUsername
--setAlias my-hub-org
```

To specify a different login URL:

```
sfdx force:auth:jwt:grant --clientid 04580y4051234051 --jwtkeyfile
/Users/jdoe/JWT/server.key --username jdoe@acdxgs0hub.org --setDefaultDevHubUsername
--setAlias my-hub-org --instanceUrl https://test.salesforce.com
```

SEE ALSO:

[Create a Private Key and Self-Signed Digital Certificate](#)

[Create a Connected App](#)

[Salesforce DX Project Configuration](#)

[Trailhead: Create Your Connected App \(Continuous Integration Using Salesforce DX Module\)](#)

Create a Private Key and Self-Signed Digital Certificate

The JWT-based authorization flow requires a digital certificate and the private key used to sign the certificate. You upload the digital certificate to the custom connected app that is also required for JWT-based authorization. You can use your own private key and certificate issued by a certification authority. Alternatively, you can use OpenSSL to create a key and a self-signed digital certificate.

This process produces two files.

- `server.key`—The private key. You specify this file when you authorize an org with the `force:auth:jwt:grant` command.
- `server.crt`—The digital certification. You upload this file when you create the connected app required by the JWT-based flow.

1. If necessary, install OpenSSL on your computer.

To check whether OpenSSL is installed on your computer, run this command.

```
$ which openssl
```

2. In Terminal or a Windows command prompt, create a directory to store the generated files, and change to the directory.

```
$ mkdir /Users/jdoe/JWT
$ cd /Users/jdoe/JWT
```

3. Generate a private key, and store it in a file called `server.key`.

```
$ openssl genrsa -des3 -passout pass:x -out server.pass.key 2048
$ openssl rsa -passin pass:x -in server.pass.key -out server.key
```

You can delete the `server.pass.key` file because you no longer need it.

4. Generate a certificate signing request using the `server.key` file. Store the certificate signing request in a file called `server.csr`. Enter information about your company when prompted.

```
$ openssl req -new -key server.key -out server.csr
```

5. Generate a self-signed digital certificate from the `server.key` and `server.csr` files. Store the certificate in a file called `server.crt`.

```
$ openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt
```

SEE ALSO:

[OpenSSL: Cryptography and SSL/TLS Tools](#)

[Create a Connected App](#)


[Authorize an Org Using the JWT-Based Flow](#)

Create a Connected App

If you use JWT-based authorization, you must create your own connected app in your Dev Hub org. You can also create a connected app for web-based authorization if you require more security than provided with our connected app. For example, you can create a connected app to set the refresh token timeout or specify IP ranges.

You create a connected app using Setup in your Dev Hub org. These steps assume that you are using Lightning Experience.

JWT-based authorization requires a digital certificate, also called a digital signature. You can use your own certificate or create a self-signed certificate using OpenSSL.


 **Note:** The steps marked *JWT only* are required only if you are creating a connected app for JWT-based authorization. They are optional for web-based authorization.

1. Log in to your Dev Hub org.
2. From Setup, enter *App Manager* in the Quick Find box to get to the Lightning Experience App Manager.
3. In the top-right corner, click **New Connected App**.

4. Update the basic information as needed, such as the connected app name and your email address.
5. Select **Enable OAuth Settings**.
6. For the callback URL, enter `http://localhost:1717/OauthRedirect`.

If port 1717 (the default) is already in use on your local machine, specify an available one instead. Make sure to also update your `sfdx-project.json` file by setting the `oauthLocalPort` property to the new port. For example, if you set the callback URL to `http://localhost:1919/OauthRedirect`:

```
"oauthLocalPort" : "1919"
```

7. (JWT only) Select **Use digital signatures**.
8. (JWT only) Click **Choose File** and upload the `server.crt` file that contains your digital certificate.
9. Add these OAuth scopes:
 - **Access and manage your data (api)**
 - **Perform requests on your behalf at any time (refresh_token, offline_access)**
 - **Provide access to your data via the Web (web)**
10. Click **Save**.
 -  **Important:** Make note of the consumer key because you need it later when you run a `force:auth` command.
11. (JWT only) Click **Manage**.
12. (JWT only) Click **Edit Policies**.
13. (JWT only) In the OAuth Policies section, select **Admin approved users are pre-authorized** for Permitted Users, and click **OK**.
14. (JWT only) Click **Save**.
15. (JWT only) Click **Manage Profiles** and then click **Manage Permission Sets**. Select the profiles and permission sets that are pre-authorized to use this connected app. Create permission sets if necessary.

SEE ALSO:

[Create a Private Key and Self-Signed Digital Certificate](#)

[Trailhead: Create Your Connected App \(Continuous Integration Using Salesforce DX Module\)](#)

[Connected Apps](#)

[Authorization](#)

Use an Existing Access Token Instead of Authorizing

When you authorize into an org using the `force:auth` commands, the Salesforce CLI takes care of generating and refreshing all tokens, such as the access token. But sometimes you want to run a few CLI commands against an existing org without going through the entire authorization process. In this case, you must provide the access token and instance URL of the org.

1. Use `force:config:set` to set the `instanceUrl` config value to the Salesforce instance that hosts the existing org to which you want to connect.

```
sfdx force:config:set instanceUrl=https://na35.salesforce.com
```

- When you run the CLI command, use the org's access token as the value for the `--targetusername` parameter rather than the org's username.

```
sfdx force:mdapi:deploy --deploydir <md-dir> --targetusername <access-token>
```

The CLI does not store the access token in its internal files. It uses it only for this CLI command run.

Authorization Information for an Org

You can view information for all orgs that you have authorized and the scratch orgs that you have created.

Use this command to view authentication information about an org.

```
sfdx force:org:display --targetusername <username>
```


If you have set a default username, you don't have to specify the `--targetusername` parameter. To display the usernames for all the active orgs that you've authorized or created, use `force:org:list`.

If you have set an alias for an org, you can specify it with the `--targetusername` parameter. This example uses the `my-scratch` alias.

```
$ sfdx force:org:display --targetusername my-scratch-org

=== Org Description
KEY                VALUE
-----
Access Token      <long-string>
Alias              my-scratch-org
Client Id         SalesforceDevelopmentExperience
Created By        joe@mydevhub.org
Created Date      2017-06-07T00:51:59.000+0000
Dev Hub Id        jdoe@fabdevhub.org
Edition           Developer
Expiration Date   2017-06-14
Id                00D9A0000008cKm
Instance Url      https://page-power-5849-dev-ed.cs46.my.salesforce.com
Org Name          Your Company
Status            Active
Username          test-apraqvkwhtml@your_company.net
```

To get more information, such as the Salesforce DX authentication URL, include the `--verbose` parameter.

 **Note:** To help prevent security breaches, the `force:org:display` output doesn't include the org's client secret or refresh token. If you need these values, perform an OAuth flow outside of the Salesforce CLI.

SEE ALSO:

[OAuth 2.0 Web Server Authentication Flow](#)

[Salesforce DX Usernames and Orgs](#)

CHAPTER 4 Scratch Orgs

In this chapter ...

- [Scratch Org Definition File](#)
- [Scratch Org Definition Configuration Values](#)
- [Create Scratch Orgs](#)
- [Supported Metadata API Types](#)
- [Salesforce DX Project Structure and Source File Format](#)
- [Push Source to the Scratch Org](#)
- [Assign a Permission Set](#)
- [Ways to Add Data to Your Scratch Org](#)
- [Pull Source from the Scratch Org to Your Project](#)
- [Generate a Password for a Scratch Org](#)
- [Manage Scratch Orgs from the Dev Hub](#)

The scratch org is a source-driven and disposable deployment of Salesforce code and metadata. A scratch org is fully configurable, allowing developers to emulate different Salesforce editions with different features and preferences. And you can share the scratch org configuration file with other team members, so you all have the same basic org in which to do your development.

Scratch orgs drive developer productivity and collaboration during the development process, and facilitate automated testing and continuous integration. You might spin up a new scratch org when you want to:

- Start a new project.
- Start a new feature branch.
- Test a new feature.
- Start automated testing.
- Perform development tasks directly in an org.
- Start from “scratch” with a fresh new org.

You can use the CLI or IDE to open your scratch org in a browser without logging in.

Scratch Org Limits and Considerations

To ensure optimal performance using Salesforce DX:

- You can create up to 50 scratch orgs per day per Dev Hub.
- You can have up to 25 active scratch orgs.
- Salesforce deletes scratch orgs and their associated active scratch org objects when a scratch org is older than 7 days.

If you want to try out Salesforce DX, you can sign up for a [Dev Hub trial org](#) with these limits:

- The Dev Hub trial org expires after 30 days.
- Your Dev Hub org gets 10 user licenses.
- You can create up to 40 scratch orgs per day per trial Dev Hub.
- You can have up to 20 active scratch orgs.
- Salesforce deletes scratch orgs and their associated scratch org objects when a scratch org is older than 7 days.

Scratch Org Definition File

The scratch org definition file is a blueprint for a scratch org. It mimics the shape of an org that you use in the development life cycle, such as sandbox, packaging, or production.

The shape of an org is determined by the collection of settings associated with it, including:

- Edition—The Salesforce edition of the scratch org, such as Developer, Enterprise, Group, or Professional.
- Add-on features—Licenses that can be added onto an edition, sometimes for an extra fee, such as multi-currency.
- Org preferences—Org and feature settings used to configure Salesforce products, such as Chatter and Communities.

For example, you can turn Chatter on or off in a scratch org by setting the `ChatterEnabled` org preference in the configuration file. Setting up different configuration files allows you to easily create scratch orgs with different shapes for testing.

```
{
  "orgName": "Acme",
  "country": "US",
  "edition": "Enterprise",
  "features": "MultiCurrency;AuthorApex",
  "orgPreferences": {
    "enabled": ["S1DesktopEnabled", "ChatterEnabled"],
    "disabled": ["SelfSetPasswordInApi"]
  }
}
```

Here are the options you can specify in the scratch org definition file:

Name	Required?	Default If Not Specified
orgName	No	Company
country	No	Dev Hub's country
username	No	<code>test-unique_identifier@orgName.net</code>
adminEmail	No	Email address of the Dev Hub user making the scratch org creation request
edition	Yes	None. Valid entries are Developer, Enterprise, Group, or Professional
language	No	Default language for the country
features	No	None
orgPreferences	No	None

⚠ Important: When indicating an `orgName`, don't use a period as the last character. For example, Acme Corp. is not a good name because it causes an undesirable extra period in the scratch org username (`scratchorg1497932510400@acmecorp..com`).

Some features, such as Communities, can require a combination of a feature and an `orgPreferences` parameter to work correctly for scratch orgs. This code snippet sets both the feature and org preference.

```
"features": "Communities",
  "orgPreferences": {
```

```
"enabled": ["NetworksEnabled"],  
...
```

You indicate the path to the scratch org configuration file when you create a scratch org with the `force:org:create` CLI command. You can name this file whatever you like and locate it anywhere the CLI can access.

If you're using a sample repo or creating a Salesforce DX project, the sample scratch org definition files are located in the config directory. You can create different configuration files for the different purposes or org shapes. For easy identification, name the file something descriptive, such as `devEdition-scratch-def.json` or `packaging-org-scratch-def.json`.

We recommend that you keep this file in your project and check it in to your version control system. For example, create a team version that you check in for all team members to use. Individual developers could also create their own local version that includes the scratch org definition parameters. Examples of these parameters include email and last name, which identify who is creating the scratch org.

Scratch Org Definition Configuration Values

Configuration values determine the shape of the scratch org.

Supported Editions

The Salesforce edition of the scratch org. Possible values are:

- Developer
- Enterprise
- Group
- Professional

Supported Features

You can enable these add-on features in a scratch org:

- API
- AuthorApex
- Communities
- ContractApprovals
- CustomerSelfService
- CustomApps
- CustomTabs
- ForceComPlatform
- MultiCurrency
- PersonAccounts
- SalesWave
- SControls
- ServiceCloud
- ServiceWave
- Sites

You can specify multiple feature values in a semi-colon delimited list.



Note: For Group and Professional Edition orgs, the AuthorApex feature is disabled by default. Enabling the AuthorApex feature lets you edit and test your Apex classes.

Supported Org Preferences

Org preferences are settings that a user can configure in the org. For example, these preferences control which Chatter, Knowledge, and Opportunities settings are enabled, among many others. These settings are enabled (or disabled) in the OrgPreferences section of the configuration file, in JSON format.

```
"OrgPreferences": {
  "enabled": ["S1DesktopEnabled", "ChatterEnabled", "IsOpportunityTeamEnabled"],
  "disabled": ["IsOrdersEnabled"]
}
```

You can set the following org preferences in the configuration file. See the *Metadata API Developer Guide* for descriptions of the supported settings.

General Settings

- AnalyticsSharingEnable
- AsyncSaveEnabled
- ChatterEnabled
- EnhancedEmailEnabled
- EventLogWaveIntegEnabled
- LoginForensicsEnabled
- NetworksEnabled
- OfflineDraftsEnabled
- PathAssistantsEnabled
- S1DesktopEnabled
- S1EncryptedStoragePref2
- S1OfflinePref
- SelfSetPasswordInApi
- SendThroughGmailPref
- SocialProfilesEnable
- Translation
- VoiceEnabled

Account Settings

- IsAccountTeamsEnabled
- ShowViewHierarchyLink

Activities Settings

- IsActivityRemindersEnabled
- IsDragAndDropSchedulingEnabled
- IsEmailTrackingEnabled
- IsGroupTasksEnabled

- IsMultidayEventsEnabled
- IsRecurringEventsEnabled
- IsRecurringTasksEnabled
- IsSidebarCalendarShortcutEnabled
- IsSimpleTaskCreateUIEnabled
- ShowEventDetailsMultiUserCalendar
- ShowHomePageHoverLinksForEvents
- ShowMyTasksHoverLinks
- ShowRequestedMeetingsOnHomePage

Contract Settings

- AutoCalculateEndDate
- IsContractHistoryTrackingEnabled
- NotifyOwnersOnContractExpiration

Entitlement Settings

- AssetLookupLimitedToActiveEntitlementsOnAccount
- AssetLookupLimitedToActiveEntitlementsOnContact
- AssetLookupLimitedToSameAccount
- AssetLookupLimitedToSameContact
- IsEntitlementsEnabled
- EntitlementLookupLimitedToActiveStatus
- EntitlementLookupLimitedToSameAccount
- EntitlementLookupLimitedToSameAsset
- EntitlementLookupLimitedToSameContact

Forecasting Settings

- IsForecastsEnabled

Ideas Settings

- IsChatterProfileEnabled
- IsIdeaThemesEnabled
- IsIdeasEnabled
- IsIdeasReputationEnabled

Knowledge Settings

- IsCreateEditOnArticlesTabEnabled
- IsExternalMediaContentEnabled
- IsKnowledgeEnabled
- ShowArticleSummariesCustomerPortal
- ShowArticleSummariesInternalApp
- ShowArticleSummariesPartnerPortal
- ShowValidationStatusField

Live Agent Settings

- IsLiveAgentEnabled

Marketing Action Settings

- IsMarketingActionEnabled

Name Settings

- IsMiddleNameEnabled
- IsNameSuffixEnabled

Opportunity Settings

- IsOpportunityTeamEnabled

Order Settings

- IsOrdersEnabled

Personal Journey Settings

- IsExactTargetForSalesforceAppsEnabled

Product Settings

- IsCascadeActivateToRelatedPricesEnabled
- IsQuantityScheduleEnabled
- IsRevenueScheduleEnabled

Quote Settings

- IsQuoteEnabled

Search Settings

- DocumentContentSearchEnabled
- OptimizeSearchForCjkEnabled
- RecentlyViewedUsersForBlankLookupEnabled
- SidebarAutoCompleteEnabled
- SidebarDropDownListEnabled
- SidebarLimitToItemsShownCheckboxEnabled
- SingleSearchResultShortcutEnabled
- SpellCorrectKnowledgeSearchEnabled

SEE ALSO:

[Salesforce Editions](#)

[Settings \(Metadata API Developer Guide\)](#)

Create Scratch Orgs

After you create the scratch org definition file, you can easily spin up a scratch org and open it directly from the command line.

Before you create a scratch org:

- Set up your Salesforce DX project
- Authorize the Dev Hub
- Create the scratch org definition file

You can create scratch orgs for different purposes, such as for feature development, for development of packages that contain a namespace, or user acceptance testing.

1. Create the scratch org.

To	Run This Command
Create a scratch org for development using a scratch org definition file	The scratch org definition defines the org type, org shape, and other options. If you are developing or updating features for a package, then your scratch org definition file probably indicates a namespace. <pre>sfdx force:org:create -f project-scratch-def.json</pre>
Specify scratch org definition values on the command line using key=value pairs	<pre>sfdx force:org:create adminEmail=me@email.com edition=Developer username=admin_user@orgname.org</pre>
Create a scratch org with an alias	Scratch org usernames are long and unintuitive. Setting an alias each time you create a scratch org is a great way to track the scratch org's purpose. And it's much easier to remember when issuing subsequent CLI commands. <pre>sfdx force:org:create -f project-scratch-def.json -a MyScratchOrg</pre>
Create a scratch org for user acceptance testing or to test installations of packages	In this case, you don't want to create a scratch org with a namespace, so you can use this command to override the namespace value in the scratch org definition file. <pre>sfdx force:org:create -f project-scratch-def.json --nonamespace</pre>
Indicate that this scratch org is the default	CLI commands that are run from within the project use the default scratch org, and you don't have to manually enter the username parameter each time. <pre>sfdx force:org:create -f project-scratch-def.json --setDefaultUsername</pre>

Indicate the path to the scratch definition file relative to your current directory. For sample repos, this file is located in the config directory.

Stdout displays two important pieces of information: the org ID and the username.

```
Successfully created scratch org: 00D3D0000000PE5UAM, username:
scratchorg1488709604@acme.com
```

If the create command times out before the scratch org is created (the default wait time is 6 minutes), you see an error. Issue this command to see if it returns the scratch org ID, which confirms the existence of the scratch org:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo WHERE
CreatedBy.Name = '<your name>' AND CreatedDate = TODAY" -u <Dev Hub org>
```

If your name is Jane Doe, and you created an alias for your Dev Hub org called DevHub, your query looks like this:

```
sfdx force:data:soql:query -q "SELECT ID, Name, Status FROM ScratchOrgInfo WHERE
CreatedBy.Name = 'Jane Doe' AND CreatedDate = TODAY" -u DevHub
```

If that doesn't work, create another scratch org and increase the timeout value using the `--wait` parameter.

2. Open the org.

```
sfdx force:org:open -u <username/alias>
```

If you want to open the scratch org in Lightning Experience or open a Visualforce page, use the `--path` parameter.

```
sfdx force:org:open --path one/one.app // opens in Lightning Experience
```

3. Push local project source to your scratch org.

SEE ALSO:

[Project Setup](#)

[Authorization](#)

[Scratch Org Definition File](#)

[Push Source to the Scratch Org](#)

Supported Metadata API Types

After changing the metadata source, you can sync the changes between the project and scratch org. We describe how to do that next. In general, we mainly support the Metadata API types that are packageable and a few non-packageable components.

We support the packageable components listed in the *ISVforce Guide* and these others:

- ActionLinkGroupTemplate
- CorsWhitelistOrigin
- CustomMetadata

We also support these non-packageable components:

- Group
- Profile

For more information, see “Metadata Types” in the *Metadata API Developer Guide*.

SEE ALSO:

[Components Available in Managed Packages \(ISVForce Guide\)](#)

[Metadata Types \(Metadata API Developer Guide\)](#)

Salesforce DX Project Structure and Source File Format

A Salesforce DX project has a specific project structure and source file format. Salesforce DX source uses a different set of files and file extensions from what you're accustomed to using Metadata API.

File Extensions

When you convert existing Metadata API source to Salesforce DX project format, we create an XML file for each bit. All files that contain XML markup now have an `.xml` extension. You can then look at your source files using an XML editor. To sync your local projects and

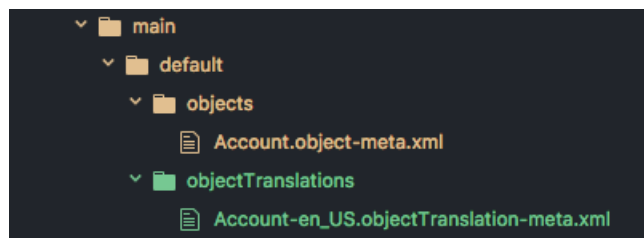
scratch orgs, Salesforce DX projects use a particular directory structure for custom objects, custom object translations, Lightning components, and documents.

For example, if you had an object called `Case.object`, Salesforce DX provides an XML version called `Case.object-meta.xml`. If you have an app call `DreamHouse.app`, we create a file called `DreamHouse.app-meta.xml`. You get the idea. For Salesforce DX projects, all source files have a companion file with the “-meta.xml” extension.

Source Transformation

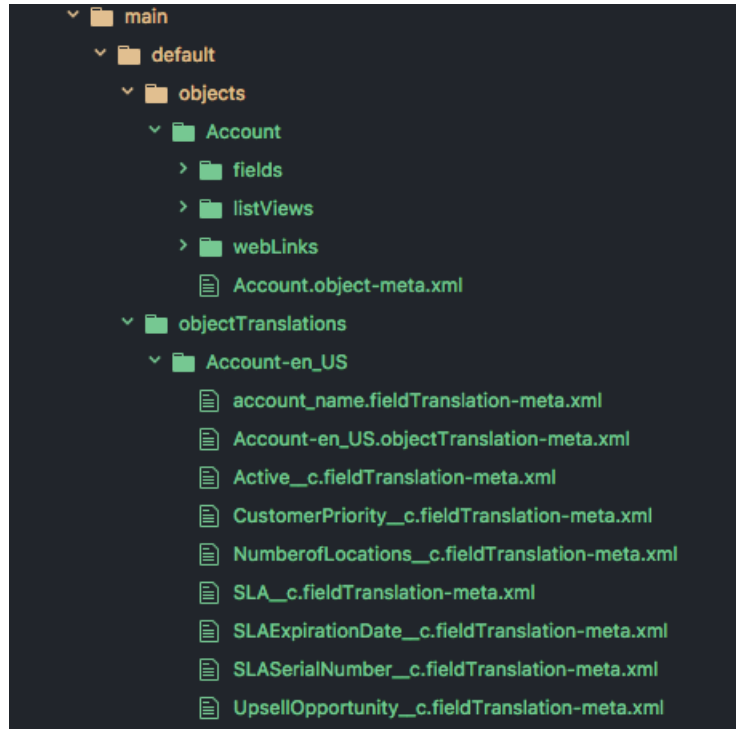
It's not uncommon for Metadata API source files to be quite large, making it difficult to find what you want. If you work on a team with other developers who make changes or updates to the same source Metadata API source file at the same time, you have to deal with merging multiple updates to the file. If you're thinking that there has to be a better way, you're right.

Before Salesforce DX, all custom objects and object translations were stored in one large Metadata API source file.



Salesforce DX solves this problem by providing a new source shape that breaks down these large source files to make them more digestible and easier to manage with a version control system.

A Salesforce DX project stores custom objects and custom object translations in intuitive subdirectories. This source structure makes it much easier to find what you want to change or update. And you can say goodbye to messy merges.



Custom Objects

When you convert your source to Salesforce DX project format, your custom objects are placed in the `<package directory>/main/default/objects` directory. Each object has its own subdirectory that reflects the type of custom object. Some parts of the custom objects are extracted into in these subdirectories:

- businessProcesses
- compactLayouts
- fields
- fieldSets
- listViews
- recordTypes
- sharingReasons
- validationRules
- webLinks

The parts of the custom object that are not extracted are placed in a file.

- For objects, `<object>.object-meta.xml`
- For fields, `<field_name>.field-meta.xml`

Custom Object Translations

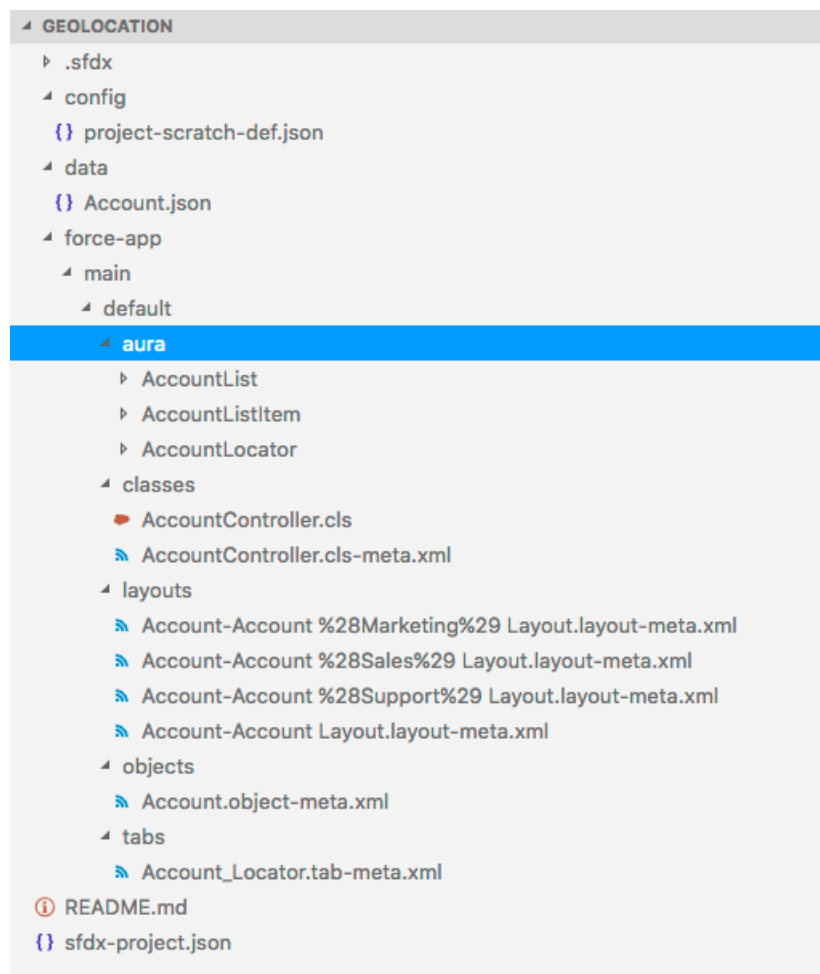
Custom object translations reside in the `<package_directory>/main/default/objectTranslations` directory, each in their own subdirectory named after the custom object translation. Custom object translations and field translations are extracted into their own files within the custom object translation's directory.

- For field names, `<field_name>.fieldTranslation-meta.xml`
- For object names, `<object_name>.objectTranslation-meta.xml`

The remaining pieces of the custom object translation are placed in a file called `<objectTranslation>.objectTranslation-meta.xml`.

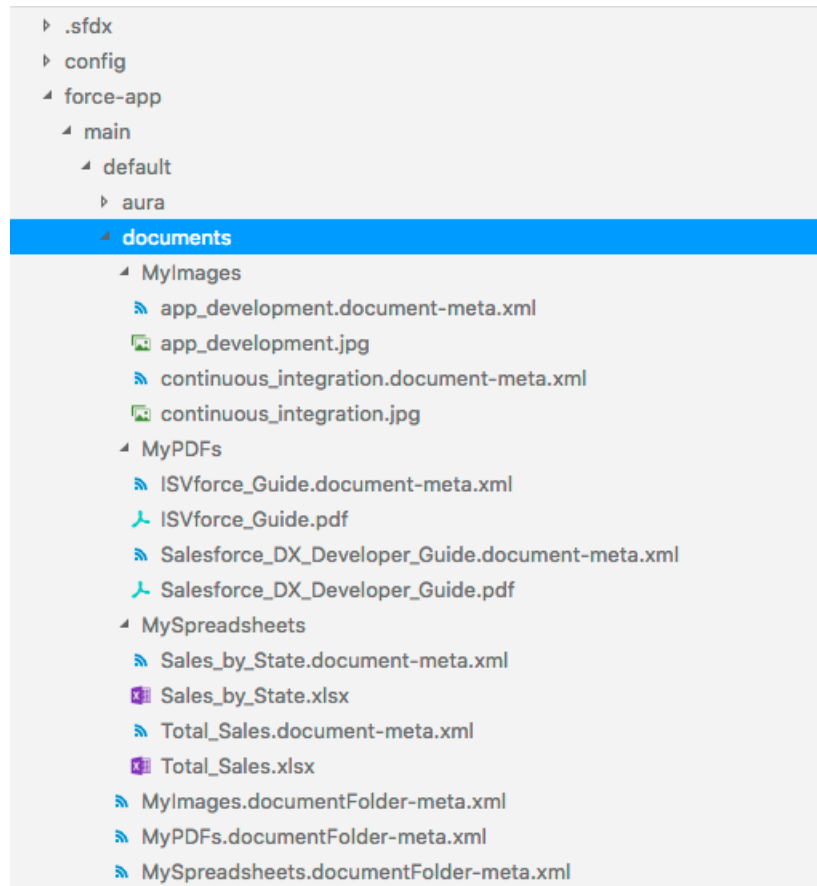
Lightning Components

Lightning bundles and components must reside in a directory named `aura` under the `<package_directory>` directory.



Documents

Documents must be inside the directories of their parent document folder. The parent document folder must be in a directory called `documents`. Each document has a corresponding Metadata API XML file that you can view with an XML editor.



Push Source to the Scratch Org

After changing the source, you can sync the changes to your scratch org by pushing the changed source to it.

The first time you push metadata to the org, all source in the folders you indicated as package directories is pushed to the scratch org to complete the initial setup. At this point, we start change-tracking locally on the file system and remotely in the scratch org to determine which metadata has changed. Let's say you pushed an Apex class to a scratch org and then decide to modify the class in the scratch org instead of your local file system. The CLI tracks in which local package directory the class was created, so when you pull it back to your project, it knows where it belongs.

 **Warning:** You can use `force:source:push` for scratch orgs only. If you're synchronizing source to another org, use the Metadata API.

During development, you change files locally in your file system and change the scratch org directly using the builders and editors that Salesforce supplies. Usually, these changes don't cause a conflict and involve unique files.

The push command doesn't handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the push command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the push command and force the changes in your project to the scratch org.

Before running the push command, you can get a list of what's new, changed, and the conflicts between your local file system and the scratch org by using `force:source:status`. This way you can choose ahead of time which version you want to keep and manually address the conflict.

Pushing Source to a Scratch Org

To push changed source to your default scratch org:

```
sfdx force:source:push
```


STATE	FULL NAME	TYPE	PROJECT PATH
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls-meta.xml
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls

To push changed source to a scratch org that's not the default, you can indicate it by its username or alias:

```
sfdx force:source:push --targetusername scratchorg148@acme.com
```

```
sfdx force:source:push -u scratchorg148@acme.com
```

```
sfdx force:source:push -u MyGroovyScratchOrg
```

 **Tip:** You can create an alias for an org using `force:alias:set`. Run `force:org:list` to display the usernames of all the scratch orgs you have created.


Selecting Files to Ignore During Push

It's likely that you have some files that you don't want to sync between the project and scratch org. You can have the push command ignore the files you indicate in `.forceignore`.

If Push Detects Warnings

If you run `force:source:push`, and warnings occur, the CLI doesn't push the source. Warnings can occur, for example, if your project source is using an outdated version. If you want to ignore these warnings and push the source to the scratch org, run:

```
sfdx force:source:push --ignorewarnings
```

 **Tip:** Although you can successfully push using this option, we recommend addressing the issues in the source files. For example, if you see a warning because a Visualforce page is using an outdated version, consider updating your page to the current version of Visualforce. This way, you can take advantage of new features and performance improvements.

If Push Detects File Conflicts

If you run `force:source:push`, and conflicts are detected, the CLI doesn't push the source.

STATE	FULL NAME	TYPE	PROJECT PATH
-------	-----------	------	--------------

Conflict	NewClass	ApexClass	/classes/CoolClass.cls-meta.xml
Conflict	NewClass	ApexClass	/classes/CoolClass.cls

Notice that you have a conflict. CoolClass exists in your scratch org but not in the local file system. In this new development paradigm, the local project is the source of truth. Consider if it makes sense to overwrite the conflict in the scratch org.

If conflicts have been detected and you want to override them, here's how you use the power of the force (overwrite) to push the source to a scratch org.

```
sfdx force:source:push --forceoverwrite
```

Next steps:

- To verify that the source was uploaded successfully to the scratch org, open the org in a browser.
- Add some sample test data.

[How to Exclude Source When Syncing or Converting](#)

When syncing metadata between your local file system and a scratch org, you often have source files you want to exclude. Similarly, you often want to exclude certain files when converting source to Salesforce DX project format. In both cases, you can exclude individual files or all files in a specific directory.

SEE ALSO:

[How to Exclude Source When Syncing or Converting](#)

[Track Changes Between the Project and Scratch Org](#)

[Track Changes Between the Project and Scratch Org](#)

[Assign a Permission Set](#)

[Ways to Add Data to Your Scratch Org](#)

How to Exclude Source When Syncing or Converting

When syncing metadata between your local file system and a scratch org, you often have source files you want to exclude. Similarly, you often want to exclude certain files when converting source to Salesforce DX project format. In both cases, you can exclude individual files or all files in a specific directory.

Use your favorite text editor to create a `.forceignore` file to specify the files or directories you want to exclude. Any source file or directory that begins with a "dot," such as `.DS_Store` or `.sfdx`, is excluded by default.

Where to Put `.forceignore`

For the `.forceignore` file to work its magic, you must put it in the proper location, depending on which command you are running.

- Add the `.forceignore` file to the root of your project for `force:source:push`.
- Add the file to the Metadata retrieve directory (with `package.xml`) for `force:mdapi:convert`.

Sample Syntax for .forceignore

The `.forceignore` file has similar functionality to `.gitignore`. Here are some options for indicating which source to exclude. All paths are relative to the project root directory.

```
# Specify a relative path to a directory from the project root
helloWorld/main/default/classes

# Specify a wildcard directory - any directory named "classes" is excluded
**classes

# Specify file extensions
**.cls
**.pdf

# Specify a specific file
helloWorld/main/default/HelloWorld.cls
```

Assign a Permission Set

After creating your scratch org and pushing the source, you must sometimes give your users access to your application, especially if your app contains custom objects.

1. If needed, create the permission set in the scratch org.
 - a. Create a scratch org.
 - b. From Setup, enter `Perm` in the Quick Find box, then select **Permission Sets**.
 - c. Click **New**.
 - d. Enter a descriptive label for the permission set, then click **Save**.
 - e. Under Apps, click **Assigned Apps > Edit**.
 - f. Under Available Apps, select your app, then click **Add** to move it to Enabled Apps.
 - g. Click **Save**.
2. Pull the permission set from the scratch org to your project.

```
sfdx force:source:pull -u <scratch org username/alias>
```

3. Assign the permission set to the org that contains the app:

```
sfdx force:user:permset:assign -n <permset_name> -u <org username/alias>
```

If a target username isn't specified, the permission set is assigned in the default scratch org using the default username.

Ways to Add Data to Your Scratch Org

Orgs for development need a small set of stock data for testing. Scratch orgs come with the same set of data as the edition on which they are based. For example, Developer Edition orgs typically include 10–15 records for key standard objects, such as Account, Contact, and Lead. These records come in handy when you're testing something like a new trigger, workflow rule, Lightning component, or Visualforce page.

Sometimes, the stock data doesn't meet your development needs. Scratch orgs have many uses, so we provide you the flexibility to add the data you need for your use cases. Apex tests generally create their own data. Therefore, if Apex tests are the only tests you're running in a scratch org, you can probably forget about data for the time being. However, other tests, such as UI, API, or user acceptance tests, do need baseline data. Make sure that you use consistent data sets when you run tests of each type.

The following sections describe the Salesforce CLI commands you can use to populate your scratch orgs. The commands you use depend on your current stage of development.

You can also use the `force:data:soql:query` CLI command to run a SOQL query against a scratch org. While the command doesn't change the data in an org, it's useful for searching or counting the data. You can also use it with other data manipulation commands.

force:data:tree Commands

The SObject Tree Save API drives the `force:data:tree` commands for exporting and importing data. The commands use JSON files to describe objects and relationships. The export command requires a SOQL query to select the data in an org that it writes to the JSON files. Rather than loading all records of each type and establishing relationships, the import command loads parents and children already in the hierarchy.

force:data:bulk Commands

Bulk API drives the `force:bulk` commands for exporting a basic data set from an org and storing that data in source control. You can then update or augment the data directly rather than in the org from where it came. The `force:data:bulk` commands use CSV files to import data files into scratch orgs or to delete sets of data that you no longer want hanging around. Use dot notation to establish child-to-parent relationships.

force:data:record Commands

Everyone's process is unique, and you don't always need the same data as your teammates. When you want to create, modify, or delete individual records quickly, use the `force:data:record:create` | `delete` | `get` | `update` commands. No data files are needed.

[Example: Export and Import Data Between Orgs](#)

Let's say you've created the perfect set of data to test your application, and it currently resides in your default scratch org. You finished coding a new feature that you want to test in a new scratch org. You create the scratch org, push your source code, and assign the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

SEE ALSO:

[SObject Tree Request Body \(REST API Developer Guide\)](#)

[Create Multiple Records \(REST API Developer Guide\)](#)

[Create Nested Records \(REST API Developer Guide\)](#)

[Salesforce Object Query Language \(SOQL\)](#)

[Sample CSV File \(Bulk API Developer Guide\)](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Example: Export and Import Data Between Orgs

Let's say you've created the perfect set of data to test your application, and it currently resides in your default scratch org. You finished coding a new feature that you want to test in a new scratch org. You create the scratch org, push your source code, and assign the needed permission sets. Now you want to populate the scratch org with your perfect set of data from the other org. How? Read on!

This use case refers to the Broker and Properties custom objects of the Salesforce DX Github DreamHouse example. It's assumed that, in the first scratch org from which you are exporting data, you've created the two objects by pushing the DreamHouse source. It's also assumed that you've assigned the permission set and populated the objects with the data. In the second scratch org, however, it's assumed that you've created the two objects and assigned the permission set but not yet populated them with data. See the README of the `sfdx-dreamhouse` GitHub example for instructions on these tasks.

1. Export the data in your default scratch org.

Use the `force:data:soql:query` command to fine-tune the SELECT query so that it returns the exact set of data you want to export. This command outputs the results to your terminal or command window, but it doesn't change the data in the org. Because the SOQL query is long, the command is broken up with backslashes for easier reading. You can still cut and paste the command into your terminal window and run it.

```
sfdx force:data:soql:query --query \  
  "SELECT Id, Name, Title__c, Phone__c, Mobile_Phone__c, \  
    Email__c, Picture__c, \  
    (SELECT Name, Address__c, City__c, State__c, Zip__c, \  
      Price__c, Title__c, Beds__c, Baths__c, Picture__c, \  
      Thumbnail__c, Description__c \  
    FROM Properties__r) \  
  FROM Broker__c"
```

2. When you're satisfied with the SELECT statement, use it to export the data into a set of JSON files.

```
sfdx force:data:tree:export --query \  
  "SELECT Id, Name, Title__c, Phone__c, Mobile_Phone__c, \  
    Email__c, Picture__c, \  
    (SELECT Name, Address__c, City__c, State__c, Zip__c, \  
      Price__c, Title__c, Beds__c, Baths__c, Picture__c, \  
      Thumbnail__c, Description__c \  
    FROM Properties__r) \  
  FROM Broker__c" \  
  --prefix export-demo --outputdir sfdx-out --plan
```

The export command writes the JSON files to the `sfdx-out` directory (in the current directory) and prefixes each file name with the string `export-demo`. The files include a plan definition file, which refers to the other files that contain the data, one for each exported object.

3. Import the data into the new scratch org by specifying the plan definition file.

```
sfdx force:data:tree:import --targetusername <test-ABC@XYZ.com> \  
  --plan sfdx-out/export-demo-Broker__c-Property__c-plan.json
```

Use the `--plan` parameter to specify the full path name of the plan execution file generated by the `force:data:tree:export` command. Plan execution file names always end in `-plan.json`.

In the previous example, you must use the `--targetusername` option because you are importing into a scratch org that is not your default. Use the `force:org:list` command to view all your scratch orgs along with their usernames and aliases. You can also use `force:config:set` to set the new scratch org as your default.

4. (Optional) Open the new scratch org and query the imported data using the Salesforce UI and SOQL.

```
sfdx force:org:open --targetusername <test-ABC@XYZ.com>
```

If you set an alias for the scratch org username, you can pass it to the `--targetusername` parameter.

```
sfdx force:org:open --targetusername <alias>
```

SEE ALSO:

[CLI Configuration Values](#)


[sfdx-dreamhouse Sample GitHub Repo](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Pull Source from the Scratch Org to Your Project

After you do an initial push, Salesforce DX tracks the changes between your local file system and your scratch org. If you change your scratch org, you usually want to pull those changes to your local project to keep both in sync.

During development, you change files locally in your file system and change the scratch org using the builders and editors that Salesforce supplies. Usually, these changes don't cause a conflict and involve unique files.

 **Important:** You can use `force:source:pull` for scratch orgs only. If you're synchronizing source to any other org, use the Metadata API (`force:mdapi:retrieve` or `force:mdapi:deploy`).

By default, only changed source is synced back to your project.

The pull command does not handle merges. Projects and scratch orgs are meant to be used by one developer. Therefore, we don't anticipate file conflicts or the need to merge. However, if the pull command detects a conflict, it terminates the operation and displays the conflict information to the terminal. You can rerun the command with the `force` option if you want to pull changes from your scratch org to the project despite any detected conflicts.

Before you run the pull command, you can get a list of what's new, changed, and any conflicts between your local file system and the scratch org by using `force:source:status`. This way you can choose ahead of time which files to keep.

To pull changed source from the scratch org to the project:

```
sfdx force:source:pull
```

You can indicate either the full scratch org username or an alias. The terminal displays the results of the pull command. This example adds two Apex classes to the scratch org. The classes are then pulled to the project in the default package directory. The pull also indicates which files have changed since the last push and if a conflict exists between a version in your local project and the scratch org.

STATE	FULL NAME	TYPE	PROJECT PATH
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls-meta.xml
Changed	MyWidgetClass	ApexClass	/classes/MyWidgetClass.cls
Changed	CoolClass	ApexClass	/classes/CoolClass.cls-meta.xml
Changed	CoolClass	ApexClass	/classes/CoolClass.cls

To pull source to the project if a conflict has been detected:

```
sfdx force:source:pull --forceoverwrite
```

SEE ALSO:

[Track Changes Between the Project and Scratch Org](#)

Generate a Password for a Scratch Org

Use the CLI to create a password for your scratch org and set it for the specified username. To change the password after you set it, you can't unset it, rerun the command.

1. Generate a password for your scratch org with this command:

```
sfdx force:user:password:generate -u <my-scratch-org>
```

You can run this command for scratch orgs only. The command outputs the generated password.

2. View the scratch org details:

```
force:org:display
```

KEY	VALUE
Access Token	008jF!AQoEr5qa2Fp1_.162DdsQE.ktgDf1N_.fVfBc_xQzO3
Alias	TempUnmanaged
Client Id	SalesforceDevelopmentExperience
Created Date	2017-04-26
Dev Hub Id	0J746000000PBp9
Edition	Developer Edition
Expiration Date	2017-05-04
Id	00D3D0000008bjF
Instance Url	https://connect-ruby-461-dev-ed.cs70.my.salesforce.com
Password	4739223f
Scratch Org	true
Username	scratchorg1493225075521@acme.com

3. Log in to the scratch org:

- a. From the `force:org:display` output, copy the value of Instance URL.

- b. Log in:

<https://login.salesforce.com/>

- c. In the login screen, click **Use Custom Domain**.

If you don't see this option, select **Log In with a Different Username**.

- d. In the Custom Domain field, paste the instance URL without the `https://`.

- e. Click **Continue**, then enter the username and password for the scratch org listed in the output of the `org:display` command.

Manage Scratch Orgs from the Dev Hub

You can view and delete your scratch orgs and their associated requests from the Dev Hub.

In the Dev Hub, `ActiveScratchOrgs` represent the scratch orgs that are currently in use. `ScratchOrgInfos` represent the requests that were used to create scratch orgs and provide historical context.

1. Log in to Dev Hub org as the System Administrator or as a user with the Salesforce DX permissions.
2. From the App Launcher, select **Active Scratch Orgs**. A list of all active scratch orgs is displayed.
To view more details about a scratch org, click the link in the Number column. To view the request that was used to create the scratch org, click the link in the Scratch Org Info column.
3. To delete an active scratch org from the Active Scratch Org list view, choose **Delete** from the dropdown.
Deleting an active scratch org does not delete the request (`ScratchOrgInfo`) that created it.
4. To view the requests that created the scratch orgs, select **Scratch Org Info** from the App Launcher.
To view more details about a request, click the link in the Number column. The details of a scratch org request include whether it's active, expired, or deleted.
5. To delete the request that was used to create a scratch org, choose **Delete** from the dropdown.
Deleting the request (`ScratchOrgInfo`) also deletes the active scratch org.

SEE ALSO:

[Add Salesforce DX Users \(Salesforce DX Setup Guide \(Beta\)\)](#)

CHAPTER 5 Development

In this chapter ...

- [Create a Lightning App and Components](#)
- [Create an Apex Class](#)
- [Track Changes Between the Project and Scratch Org](#)
- [Testing](#)

After you import some test data, you've completed the process of setting up your project. Now, you're ready to start the development process.

Create Source Files from the CLI

To add source files from the CLI, make sure that you're working in an appropriate directory. For example, if your package directory is called `force-app`, create Apex classes in `force-app/main/default/classes`. You can organize your source as you want underneath each package directory except for documents, custom objects, and custom object translations. Also, your Lightning components must be in the `aura` directory.

Execute one of these commands.

- `apex:class:create`
- `lightning:app:create`
- `lightning:component:create`
- `lightning:event:create`
- `lightning:interface:create`
- `visualforce:component:create`
- `visualforce:page:create`

Consider using these two powerful optional flags:

Option	Description
<code>-d, --outputdir</code>	The directory for saving the created files. If you don't indicate a directory, your source is added to the current folder.
<code>-t, --template</code>	Template used for the file creation.

 **Tip:** If you want to know more information about a command, run it with the `--help` option. For example, `sfdx apex:class:create --help`.

Edit Source Files

Use your favorite code editor to edit Apex classes, Visualforce pages and components, and Lightning bundles in your project. You can also make edits in your default scratch org and then use `force:source:pull` to pull those changes down to your project. For Lightning pages (FlexiPage

files) that are already in your scratch org, use the shortcut to open Lightning App Builder in a scratch org from your default browser. Lightning Pages are stored in the `flexipages` directory.

To edit a FlexiPage in your default browser—for example, to edit the `Property_Record_Page` source—execute this command.

```
sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml
```

If you want to generate a URL that loads the `.flexipage-meta.xml` file in Lightning App Builder but does not launch your browser, use the `--urlonly` flag.

```
sfdx force:source:open -f Property_Record_Page.flexipage-meta.xml -r
```

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

Create a Lightning App and Components

To create Lightning apps and components from the CLI, you must have an `aura` directory in your Salesforce DX project.

1. In `<app_dir>/main/default`, create the `aura` directory.
2. Change to the `aura` directory.
3. In the `aura` directory, create a Lightning app or a Lightning component.

```
sfdx force:lightning:app:create -n mylightningapp
```

```
sfdx force:lightning:component:create -n mylightningcomp
```

Create an Apex Class

You can create Apex classes from the CLI.

1. If the `classes` directory doesn't exist in `<app_dir>/main/default`, create it.
2. In the `classes` directory, create the class.

```
sfdx force:apex:class:create -n myclass
```

Track Changes Between the Project and Scratch Org

When you start developing, you can change local files in your project directory or remotely in your scratch org. Before you push local changes to the scratch org or pull remote changes to the local Salesforce DX project, it's helpful to see what changes you've made.

1. To view the status of local or remote files:

```
sfdx force:source:status
```

STATE	FULL NAME	TYPE	PROJECT PATH
Local Deleted	MyClass ApexClass		/MyClass.cls-meta.xml
Local Deleted	MyClass ApexClass		/MyClass.cls
Local Add	OtherClass ApexClass		/OtherClass.cls-meta.xml
Local Add	OtherClass ApexClass		/OtherClass.cls
Local Add	Event QuickAction		/Event.quickAction-meta.xml
Remote Deleted	MyWidgetClass ApexClass		/MyWidgetClass.cls-meta.xml
Remote Deleted	MyWidgetClass ApexClass		/MyWidgetClass.cls
Remote Changed (Conflict)	NewClass ApexClass		/NewClass.cls-meta.xml
Remote Changed (Conflict)	NewClass ApexClass		/NewClass.cls

Testing

When you're ready to test changes to your Force.com app source code, you can run Apex tests from the Salesforce DX CLI. The command runs Apex tests in your scratch org.

You can also execute the CLI command for running Apex tests (`force:apex:test:run`) from within third-party continuous integration tools, such as Jenkins.

To run Apex tests from the command line:

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

This command runs all Apex tests in the scratch org asynchronously and then outputs a job ID. Pass the ID to the `force:apex:test:report` command to view the results. The results include the outcome of individual tests, how long each test ran, and the overall pass and fail rate.

```
sfdx force:apex:test:report --testrunid 7074C00000988ax
```

Use the `--resultformat` parameter to run the tests synchronously. The command waits to display the test results until all tests have completed.

```
sfdx force:apex:test:run --resultformat human
```

Use parameters to list the test classes or suites to run, specify the output format, view code coverage results, and more. For example, the following command runs the `TestA` and `TestB` tests, provides results in Test Anything Protocol (TAP) format, and requests code coverage results.

```
sfdx force:apex:test:run --classnames TestA,TestB --resultformat tap --codecoverage
```

SEE ALSO:

[Test Anything Protocol \(TAP\)](#)

[Salesforce CLI Command Reference \(Beta\)](#)

CHAPTER 6 Build and Release Your App

In this chapter ...

- [Build and Release Your App with Managed Packages](#)
- [View Information About a Package](#)
- [Build and Release Your App with Metadata API](#)

When you finish writing your code, the next step is to deploy it. Your destination depends on whether you're supporting internal customers through your production environment or you're an ISV building for AppExchange. For AppExchange, you build a managed package. If you're releasing an app within your own org, you can also use the Metadata API. Whichever path you're on, we've got you covered.

If you're an ISV, you want to build a managed package. A managed package is a bundle of components that make up an application or piece of functionality. You can use a managed package to protect intellectual property because the source code of many components is not available through the package. The managed package is distributed as an app, so you can also roll out upgrades to the package. Managed packages are a great way to release applications for sale and to support licensing for your features.

When you're working with your production org, you create a .zip file of metadata components and deploy them through Metadata API. The .zip file contains:

- A package manifest (package.xml) that lists what to retrieve or deploy
- One or more XML components organized into folders

If you don't have the package source already in the SFDX format, you can retrieve it from the org and convert it using the CLI.

Build and Release Your App with Managed Packages

If you developed and tested your app, you're well on your way to releasing it. Luckily, when it's time to build and release an app as a managed package, you've got options. You can package an app you developed from scratch. If you're experimenting, you can also build the sample app from Salesforce and emulate the release process.

Working with a package is an iterative process. You typically retrieve, convert, and deploy source multiple times as you create scratch orgs, test, and update the package components.

Chances are, you already have a namespace and package defined in your packaging org. If not, run this command to open the packaging org in your browser.

```
sfdx force:org:open --targetusername me@my.org --path one/one.app#/setup/Package/home
```

In the Salesforce UI, you can define a namespace and a package. Each packaging org can have a single managed package and 1 namespace. Be sure to link the namespace to your Dev Hub org.

[Packaging Checklist](#)

Ready to deploy your packaging metadata and start creating a package?

[Deploy the Package Metadata to the Packaging Org](#)

Before you deploy the package metadata into your packaging org, you convert the format of your source so that it's readable by the Metadata API.

[Create a Beta Version of Your App](#)

Test your app in a scratch org, or share the app for evaluation by creating a beta version.

[Install the Package in a Target Org](#)

After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

[Create a Managed Package Version of Your App](#)

After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

SEE ALSO:

[ISVforce Guide](#)

[Link a Namespace to the Dev Hub Org](#)

[Retrieve Source from an Existing Managed Package](#)

Packaging Checklist

Ready to deploy your packaging metadata and start creating a package?

Take a few minutes to verify that you covered the items in this checklist and you're good to go.

1. Link the namespace of each package you want to work with to the Dev Hub org.
2. Pull the metadata of the package from your version control system to a local project.
3. Update the config files, if needed.

For example, to work with managed packages, `sfdx-project.json` must include the namespace.

```
"namespace": "acme_example",
```

4. (Optional) Create an alias for each of the orgs you want to work with.

If you haven't yet created an alias for each org you work with, consider doing that now. Using aliases is an easy way to switch between orgs when you're working in the CLI.

5. Authenticate the Dev Hub org.
6. Create a scratch org.

Recall that a scratch org is different than a sandbox org. You specify the org shape using `project-scratch.json`. To create a scratch org and set it as the `defaultusername` org, run this command from the project directory.

```
sfdx force:org:create -s -f config/project-scratch-def.json
```

7. Push source to the scratch org.
8. Update source in the scratch org as needed.
9. Pull the source from the scratch org if you used declarative tools to make changes there.

With these steps complete, you're ready to deploy your package metadata to the packaging org.

SEE ALSO:

[Sample Repository on GitHub](#)

[Authorization](#)

[Create Scratch Orgs](#)

[Push Source to the Scratch Org](#)

Deploy the Package Metadata to the Packaging Org

Before you deploy the package metadata into your packaging org, you convert the format of your source so that it's readable by the Metadata API.

It's likely that you have some files that you don't want to convert to Metadata API format. Create a `.forceignore` file to indicate which files to ignore.

1. Convert the source from Salesforce DX format to the Metadata API format.

```
sfdx force:source:convert --outputdir mdapi_output_dir --packagename managed_pkg_name
```

If the output directory doesn't exist, it's created. Be sure to include the `--packagename` so that the converted metadata is added to the managed package in your packaging org.

2. Review the contents of the output directory.

```
ls -lR mdapi_output_dir
```

3. Authenticate the packaging org, if needed. This example uses an OAuth client ID but you can also specify the org with an alias.

```
sfdx force:auth:web:login --clientid oauth_client_id
```

4. Deploy the package metadata back to the packaging org.

```
sfdx force:mdapi:deploy --deploy_dir mdapi_output_dir --targetusername me@example.com
```

The `--targetusername` can be the username or an alias. You can use other options like `--wait` to specify the number of minutes to wait. The `--zipfile` parameter lets you provide the path to a zip file that contains your metadata. Don't run tests at the same time as you deploy the metadata, though. You can run tests during the package upload process.

A message displays the job ID for the deployment.

5. Check the status of the deployment.

```
sfdx force:mdapi:deploy -u username -i jobid
```

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

[How to Exclude Source When Syncing or Converting](#)

Create a Beta Version of Your App

Test your app in a scratch org, or share the app for evaluation by creating a beta version.

If you specified the package name when you converted source to Metadata API format, both the changed and new components are automatically added to the package. Including the package name in that stage of the process lets you take full advantage of the Salesforce DX end-to-end automation.

If, for some reason, you don't want to include new components, you have two choices. You can omit the package name when you convert source or remove components from the package in the Salesforce UI before you create the package version.

Create the beta version of a managed package by running the commands against your packaging org, not the Dev Hub org.

1. Ensure that you've authorized the packaging org.

```
sfdx force:auth:web:login --targetusername me@example.com
```


2. Create the beta version of the package.

```
sfdx force:package1:version:create --packageid package_id --name package_version_name
```

If you want to protect the package with an installation key, add it now or when you create the released version of your package. The `--installationkey` supplied from the CLI is equivalent to the Password field that you see when working with packages through the Salesforce user interface. When you include a value for `--installationkey`, you or a subscriber must supply the key before you can install the package in a target org.

You're now ready to create a scratch org and install the package there for testing. By default, the `create` command generates a beta version of your managed package.

Later, when you're ready to create the Managed - Released version of your package, include the `-m` (`--managedreleased true`) parameter.

-  **Note:** After you create a managed-released version of your package, many properties of the components added to the package are no longer editable. Refer to the *ISVforce Guide* to understand the differences between beta and managed-released versions of your package.

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

[ISVforce Guide](#)

[Link a Namespace to the Dev Hub Org](#)

Install the Package in a Target Org

After you create a package with the CLI, install the package in a target org. You can install the package in any org you can authenticate, including a scratch org.

If you want to create a scratch org and set it as the `defaultusername` org, run this command from the project directory.

```
sfdx force:org:create -s -f config/project-scratch-def.json
```

To locate the ID of the package version to install, first run `force:package1:version:list`.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx000000069oAAA	033xx00000007coAAA	r00	1.0.0	Released	1
04txx000000069tAAA	033xx00000007coAAA	r01	1.1.0	Released	1
04txx000000069uAAA	033xx00000007coAAA	r02	1.2.0	Released	1
04txx000000069yAAA	033xx00000007coAAA	r03	1.3.0	Released	1
04txx000000069zAAA	033xx00000007coAAA	r04	1.4.0	Released	1

You can then copy the package version ID you want to install. For example, the ID `04txx000000069zAAA` is for version 1.4.0.

1. Install the package. You supply the package version ID, which starts with `04t`, in the required `--packageid` parameter.

```
sfdx force:package:install --packageid 04txx000000069zAAA
```

If you've set a default target org, the package is installed there. You can specify a different target org with the `--targetusername` parameter. If the package is protected by an installation key, supply the key with the `--installationkey` parameter.

To uninstall a package, open the target org and choose **Setup**. On the Installed Packages page, locate the package and choose **Uninstall**.

SEE ALSO:

[ISVforce Guide](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Create a Managed Package Version of Your App

After your testing is done, your app is almost ready to be published in your enterprise or on AppExchange. Generate a new managed package version in your Dev Hub org.

Ensure that you've authorized the packaging org and can view the existing package versions.

```
sfdx force:auth:web:login --instanceurl https://test.salesforce.com --setdefaultusername org_alias
```


View the existing package versions for a specific package to get the ID for the version you want to install.

```
sfdx force:package1:version:list --packageid 033...
```

To view details for all packages in the packaging org, run the command with no parameters.

More than 1 beta package can use the same version number. However, you can use each version number for only one *managed* package version. You can specify major or minor version numbers.

You can also include URLs for a post-installation script and release notes. Before you create a managed package, make sure that you've configured your developer settings, including the namespace prefix.

 **Note:** After you create a managed package version, you can't change some attributes of Salesforce components used in the package. See the *ISVforce Guide* has information on editable components.

1. Create the managed package. Include the `--managedreleased` parameter.

```
sfdx force:package1:version:create --packageid 033xx0000007oi --name "Spring 17"
--description "Spring 17 Release" --version 3.2 --managedreleased
```

You can use other options like `--wait` to specify the number of minutes to wait.

To protect the package with an installation key, include a value for `--installationkey`. Then, you or a subscriber must supply the key before you can install the package in a target org.

After the managed package version is created, you can retrieve the new package version ID using `force:package1:version:list`.

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

[ISVforce Guide](#)

[Link a Namespace to the Dev Hub Org](#)

View Information About a Package

View the details about a specific package version, including its metadata package ID, package name, release state, and build number.

1. From the project directory, run this command, supplying a package version ID.
`force:package1:version:display -i 04txx00000069yAAA`
 The output is similar to this example.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx00000069yAAA	033xx0000007coAAA	r03	1.3.0	Released	1
04txx00000069yAAA	033xx00000011coAAA	r03	1.4.0	Released	1

[View All Package Versions in the Org](#)

View the details about all package versions in the org.

[Package IDs](#)

When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

View All Package Versions in the Org

View the details about all package versions in the org.

1. From the project directory, run the `list` command.
`force:package1:version:list`

The output is similar to this example. When you view the package versions, the list shows a single package for multiple package versions.

METADATAPACKAGEVERSIONID	METADATAPACKAGEID	NAME	VERSION	RELEASESTATE	BUILDNUMBER
04txx000000069oAAA	033xx00000007coAAA	r00	1.0.0	Released	1
04txx000000069tAAA	033xx00000007coAAA	r01	1.1.0	Released	1
04txx000000069uAAA	033xx00000007coAAA	r02	1.2.0	Released	1
04txx000000069yAAA	033xx00000007coAAA	r03	1.3.0	Released	1
04txx000000069zAAA	033xx00000007coAAA	r04	1.4.0	Released	1

SEE ALSO:

[Salesforce CLI Command Reference \(Beta\)](#)

Package IDs

When you work with packages using the CLI, the package IDs refer either to a unique package or a unique package version.

The relationship of package version to package is one-to-many.

ID Example	Description	Used Where
033xx00000007oi	Metadata Package ID	Generated when you create a package. A single package can have one or more associated package version IDs. The package ID remains the same, whether it has a corresponding beta or released package version.
04tA000000081MX	Metadata Package Version ID	Generated when you create a package version.

Build and Release Your App with Metadata API

After developing and testing your app in a scratch org, you use the Metadata API to deploy the app to a sandbox. The sandbox mimics the release activity, commands, and process you plan to execute when you release your app to the production org.

To deploy Apex to production, unit tests of your Apex code must meet coverage requirements. Code coverage indicates how many executable lines of code in your classes and triggers are covered by your test methods. Write test methods to test your triggers and classes, and then run those tests to generate code coverage information.

Code coverage requirements must be met when you release an app to a production org. If you run tests with the release and review the results, you can meet these requirements.

If you don't specify a test level when initiating a deployment, the default test execution behavior depends on the contents of your deployment package.

- If your deployment package contains Apex classes or triggers, when you deploy to production, all tests are executed, except tests that originate from a managed package.
- If your package doesn't contain Apex components, no tests are run by default.

You can run tests for a deployment of non-Apex components. You can override the default test execution behavior by setting the test level in your deployment options. Test levels are enforced regardless of the types of components present in your deployment package. We recommend that you run all local tests in your development environment, such as a sandbox, before deploying to production. Running tests in your development environment reduces the number of tests required in a production deployment.

[Deploy Changes to a Sandbox for Validation](#)

When you're ready to validate your source, convert your Salesforce DX source to Metadata API source format. You can then deploy to a sandbox.

[Release Your App to Production](#)

After you convert your source to Metadata API format and package metadata from one org, you can release your app in a different org. You can specify tests to run after deployment and indicate whether to roll back the deployment if there are errors.

SEE ALSO:


[Metadata API Developer Guide](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Deploy Changes to a Sandbox for Validation

When you're ready to validate your source, convert your Salesforce DX source to Metadata API source format. You can then deploy to a sandbox.

You can deploy or retrieve up to 10,000 files at once. The maximum size of the deployed or retrieved .zip file is 400 MB (39 MB compressed). If either limit is exceeded, the operation fails.

-  **Note:** You can increase the efficiency of your sandbox and production deployments by using tests you've already done in the scratch org. Run only the tests that are required, such as tests for Apex classes and triggers that change for the deployment. To run only specified tests when you deploy, set `-l` to `RunSpecifiedTests` and use `-r` to specify a comma-separated list of tests for deployment-specific changes to your Apex code.

```
sfdx force:mdapi:deploy -d mdapi_output_dir/ -u "sandbox_username" -l RunSpecifiedTests
-r test1,test2,test3,test4
```

The username can be the org username or an alias.

1. To ensure that your project is up to date, synchronize the source in your local file system with your development scratch org.

```
sfdx force:source:pull
```


2. From the project, create directory for your source and convert the source to Metadata API format.

- a. Create a directory for the converted source.

```
mkdir mdapi_output_dir
```

- b. Convert the source to Metadata API format, and put the converted source in the output directory that you created.

```
sfdx force:source:convert -d mdapi_output_dir/ --packagename package_name
```

-  **Note:** The `source:convert` command creates the package manifest file, `package.xml`. The `package.xml` manifest file lists the metadata to retrieve or deploy. Creating a `package.xml` file that you can modify gives you flexibility and control over the components that you're retrieving or deploying.

- c. List the contents of the output directory to confirm that it's what you expected.

```
ls -lR mdapi_output_dir/
```

3. Deploy the metadata from the directory to the sandbox, specifying deployment-specific tests as needed.

```
sfdx force:mdapi:deploy -d mdapioutput_dir/ -u "sandbox_username" -l RunSpecifiedTests -r test1,test2,test3,test4
```

Messages similar to the following display.

```
=== Status
Status: Pending
jobid: 0AfB0000009SvyoKAC
Component errors: 0
Components deployed: 0
Components total: 0
Tests errors: 0
Tests completed: 0
Tests total: 0
```

If the deployment job wait time is 1 minute or more, the status messages update every 30 seconds.

4. If your deployment exceeds the wait time before it completes, use the `jobid` value to check the deployment status. The default wait time is 0 minutes. Use the `--wait` parameter to specify a longer wait time.

For example, to check a deployment job and add 5 more minutes to the wait time:

```
sfdx force:mdapi:deploy -u "sandbox_username" -i 0AfB0000009SvyoKAC -w 5
```

The sandbox username can be the org username or an alias.

SEE ALSO:


[Metadata API Developer Guide](#)

[Salesforce CLI Command Reference \(Beta\)](#)

Release Your App to Production

After you convert your source to Metadata API format and package metadata from one org, you can release your app in a different org. You can specify tests to run after deployment and indicate whether to roll back the deployment if there are errors.

You can deploy or retrieve up to 10,000 files at once. The maximum size of the deployed or retrieved .zip file is 400 MB (39 MB compressed). If either limit is exceeded, the operation fails.

-  **Note:** To support the needs of continuous integration and automated systems, the `--rollbackonerror` parameter of the `force:mdapi:deploy` command defaults to `true`.

1. To release your .zip package of metadata to the target org, enter the following command with a list of the required tests.

```
sfdx force:mdapi:deploy -d mdapi_output_dir/ -u "target_username" -l RunSpecifiedTests -r test1,test2,test3,test4
```

The username can be the org username or an alias.

SEE ALSO:

[Metadata API Developer Guide](#)

[Salesforce CLI Command Reference \(Beta\)](#)

CHAPTER 7 Continuous Integration

In this chapter ...

- [Continuous Integration Using Jenkins](#)
- [Continuous Integration with Travis CI](#)

Continuous integration (CI) is a software development practice in which developers regularly integrate their code changes into a source code repository. To ensure that the new code does not introduce bugs, automated builds and tests run before or after developers check in their changes.

Many third-party CI tools are available for you to choose from. Salesforce DX easily integrates into these tools so that you can set up continuous integration for your Force.com applications.

Continuous Integration Using Jenkins

Jenkins is an open-source, extensible automation server for implementing continuous integration and continuous delivery. You can easily integrate Salesforce DX into the Jenkins framework to automate testing of Force.com applications against scratch orgs.

To integrate Jenkins, we assume:

- You are familiar with how Jenkins works. You can configure and use Jenkins in many ways. We focus on integrating Salesforce DX into Jenkins multibranch pipelines.
- The computer on which the Jenkins server is running has access to your version control system and to the repository that contains your Force.com application.

[Configure Your Environment for Jenkins](#)

Before integrating Salesforce DX into your existing Jenkins framework, configure your Jenkins environment.

[Jenkinsfile Walkthrough](#)

The simple `Jenkinsfile` in the `sfdx-dreamhouse` sample Github repo shows how to integrate Salesforce DX into a Jenkins job. The sample uses Jenkins multibranch pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Force.com applications. The walkthrough highlights the Salesforce DX CLI commands to create a scratch org, upload your code, and run your tests.

SEE ALSO:

[Jenkins](#)

[Pipeline-as-code with Multibranch Workflows in Jenkins](#)

Configure Your Environment for Jenkins

Before integrating Salesforce DX into your existing Jenkins framework, configure your Jenkins environment.

1. In your Dev Hub org, [create a connected app](#) as described by the JWT-based authorization flow. This step includes obtaining or [creating a private key and digital certificate](#).

Make note of your consumer key (sometimes called a client ID) when you save the connected app. You need the consumer key to set up your Jenkins environment. Also have available the private key file used to sign the digital certificate.

2. On the computer that is running the Jenkins server, do the following.
 - a. Download and install the Salesforce DX CLI.
 - b. Store the private key file as a Jenkins Secret File using the [Jenkins Admin Credentials interface](#). Make note of the new entry's ID. You later reference this Credentials entry in your `Jenkinsfile`.
 - c. Set the following variables in your Jenkins environment.
 - `HUB_ORG_DH`—The username for the Dev Hub org, such as `juliet.capulet@myenvhub.com`.
 - `SFDC_HOST_DH`—The login URL of the Salesforce instance that is hosting the Dev Hub org. The default is `https://login.salesforce.com`
 - `CONNECTED_APP_CONSUMER_KEY_DH`—The consumer key that was returned after you created a connected app in your Dev Hub org.
 - `JWT_CRED_ID_DH`—The credentials ID for the private key file that you stored in the Jenkins Admin Credentials interface.

The names for these environment variables are just suggestions. You can use any name as long as you specify it in the `Jenkinsfile`.

3. Set up your Salesforce DX project so that you can create a scratch org.
4. (Optional) Install the Custom Tools Plugin into your Jenkins console, and create a custom tool that references the Salesforce CLI. The Jenkins walkthrough assumes that you created a custom tool named `toolbelt` in the `/usr/local/bin` directory, which is the directory in which the Salesforce CLI is installed.

SEE ALSO:

[Authorize an Org Using the JWT-Based Flow](#)

[Salesforce DX Setup Guide \(Beta\)](#)

[Jenkins: Credentials Binding Plugin](#)

[Project Setup](#)

Jenkinsfile Walkthrough

The simple `Jenkinsfile` in the `sfdx-dreamhouse` sample Github repo shows how to integrate Salesforce DX into a Jenkins job. The sample uses Jenkins multibranch pipelines. Every Jenkins setup is different. This walkthrough describes one of the ways to automate testing of your Force.com applications. The walkthrough highlights the Salesforce DX CLI commands to create a scratch org, upload your code, and run your tests.

We assume that you are familiar with the structure of the `Jenkinsfile`, Jenkins Pipeline DSL, and the Groovy programming language. This walkthrough focuses solely on Salesforce DX information. See the Salesforce DX Command Reference regarding the commands used.

This Salesforce DX workflow most closely corresponds to `Jenkinsfile` stages.

- [Define Variables](#)
- [Check Out the Source Code](#)
- [Wrap All Stages in a `withCredentials` Command](#)
- [Authorize Your Dev Hub Org and Create a Scratch Org](#)
- [Push Source and Assign a Permission Set](#)
- [Run Apex Tests](#)
- [Delete the Scratch Org](#)

Define Variables

Use the `def` keyword to define the variables required by the Salesforce DX CLI commands. Assign each variable the corresponding environment variable that you previously set in your Jenkins environment.

```
def HUB_ORG=env.HUB_ORG_DH
def SFDC_HOST = env.SFDC_HOST_DH
def JWT_KEY_CRED_ID = env.JWT_CRED_ID_DH
def CONNECTED_APP_CONSUMER_KEY=env.CONNECTED_APP_CONSUMER_KEY_DH
```

Define the `SFDC_USERNAME` variable, but don't set its value. You do that later.

```
def SFDC_USERNAME
```

Although not required, we assume you've used the Jenkins Global Tool Configuration to create the `toolbelt` custom tool that points to the CLI installation directory. In your `Jenkinsfile`, use the `tool` command to set the value of the `toolbelt` variable to this custom tool.

```
def toolbelt = tool 'toolbelt'
```

You can now reference the Salesforce CLI executable in the `Jenkinsfile` using `${toolbelt}/sfdx`.

Check Out the Source Code

Before testing your code, get the appropriate version or branch from your version control system (VCS) repository. In this example, we use the `checkout scm` Jenkins command. We assume that the Jenkins administrator has already configured the environment to access the correct VCS repository and check out the correct branch.

```
stage('checkout source') {
    // when running in multi-branch job, one must issue this command
    checkout scm
}
```

Wrap All Stages in a withCredentials Command

You previously stored the JWT private key file as a Jenkins Secret File using the Credentials interface. Therefore, you must use the `withCredentials` command in the body of the `Jenkinsfile` to access the secret file. The `withCredentials` command lets you name a credential entry, which is then extracted from the credential store and provided to the enclosed code through a variable. When using `withCredentials`, put all stages within its code block.

This example stores the credential ID for the JWT key file in the variable `JWT_KEY_CRED_ID`. You defined `JWT_KEY_CRED_ID` earlier and set it to its corresponding environment variable. The `withCredentials` command fetches the contents of the secret file from the credential store and places the contents in a temporary location. The location is stored in the variable `jwt_key_file`. You use the `jwt_key_file` variable with the `force:auth:jwt` command to specify the private key securely.

```
withCredentials([file(credentialsId: JWT_KEY_CRED_ID, variable: 'jwt_key_file')]) {
    # all stages will go here
}
```

Authorize Your Dev Hub Org and Create a Scratch Org

The `sfdx-dreamhouse` example uses one stage to authorize the Dev Hub org and create a scratch org.

```
stage('Create Scratch Org') {

    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:auth:jwt:grant --clientid
${CONNECTED_APP_CONSUMER_KEY} --username ${HUB_ORG} --jwtkeyfile ${jwt_key_file}
--setdefaultdevhubusername --instanceurl ${SFDC_HOST}"
    if (rc != 0) { error 'hub org authorization failed' }

    // need to pull out assigned username
    rmsg = sh returnStdout: true, script: "${toolbelt}/sfdx force:org:create --definitionfile
config/project-scratch-def.json --json --setdefaultusername"
    printf rmsg
    def jsonSlurper = new JsonSlurperClassic()
    def robj = jsonSlurper.parseText(rmsg)
    if (robj.status != "ok") { error 'org creation failed: ' + robj.message }
```

```

SFDC_USERNAME=robject.username
robject = null
}

```

Use the `force:auth:jwt:grant` CLI command to authorize your Dev Hub org.

You are required to run this step only once, but we suggest you add it to your `Jenkinsfile` and authorize each time you run the Jenkins job. This way you're always sure that the Jenkins job is not aborted due to lack of authorization. There is typically little harm in authorizing multiple times, although keep in mind that the API call limit for your scratch org's edition still applies.

Use the parameters of the `force:auth:jwt:grant` command to provide information about the Dev Hub org that you are authorizing. The values for the `--clientid`, `--username`, and `--instanceurl` parameters are the `CONNECTED_APP_CONSUMER_KEY`, `HUB_ORG`, and `SFDC_HOST` environment variables you previously defined, respectively. The value of the `--jwtkeyfile` parameter is the `jwt_key_file` variable that you set in the previous section using the `withCredentials` command. The `--setdefaultdevhubusername` parameter specifies that this `HUB_ORG` is the default Dev Hub org for creating scratch orgs.

Use the `force:org:create` CLI command to create a scratch org. In the example, the CLI command uses the `config/project-scratch-def.json` file (relative to the project directory) to create the scratch org. The `--json` parameter specifies that the output be in JSON format. The `--setdefaultusername` parameter sets the new scratch org as the default.

The Groovy code that parses the JSON output of the `force:org:create` command extracts the username that was auto-generated as part of the org creation. This username, stored in the `SFDC_USERNAME` variable, is used with the CLI commands that push source, assign a permission set, and so on.

Push Source and Assign a Permission Set

Let's populate your new scratch org with metadata. This example uses the `force:source:push` command to upload your source to the org. The source includes all the pieces that make up your Salesforce application: Apex classes and test classes, permission sets, layouts, triggers, custom objects, and so on.

```

stage('Push To Test Org') {

    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:source:push --targetusername
${SFDC_USERNAME}"
    if (rc != 0) {
        error 'push all failed'
    }
    // assign permset
    rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:user:permset:assign
--targetusername ${SFDC_USERNAME} --permsetname DreamHouse"
    if (rc != 0) {
        error 'push all failed'
    }
}

```

Recall the `SFDC_USERNAME` variable that contains the auto-generated username that was output by the `force:org:create` command in an earlier stage. The code uses this variable as the argument to the `--targetusername` parameter to specify the username for the new scratch org.

The `force:source:push` command pushes all the Force.com-related files that it finds in your project. Add a `.forceignore` file to your repository to list the files that you do not want pushed to the org.

After pushing the metadata, the example uses the `force:user:permset:assign` command to assign a permission set (named `DreamHouse`) to the `SFDC_USERNAME` user. The XML file that describes this permission set was uploaded to the org as part of the push.

Run Apex Tests

Now that your source code and test source have been pushed to the scratch org, run the `force:apex:test:run` command to run Apex tests.

```
stage('Run Apex Test') {
    sh "mkdir -p ${RUN_ARTIFACT_DIR}"
    timeout(time: 120, unit: 'SECONDS') {
        rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:apex:test:run --testlevel
RunLocalTests --outputdir ${RUN_ARTIFACT_DIR} --resultformat tap --targetusername
${SFDC_USERNAME}"
        if (rc != 0) {
            error 'apex test run failed'
        }
    }
}
```

You can specify various parameters to the `force:apex:test:run` CLI command. In the example:

- The `--testlevel RunLocalTests` option runs all tests in the scratch org, except tests that originate from installed managed packages. You can also specify `RunSpecifiedTests` to run only certain Apex tests or suites or `RunAllTestsInOrg` to run all tests in the org.
- The `--outputdir` option uses the `RUN_ARTIFACT_DIR` variable to specify the directory into which the test results are written. Test results are produced in JUnit and JSON formats.
- The `--resultformat tap` option specifies that the command output is in Test Anything Protocol (TAP) format. The test results that are written to a file are still in JUnit and JSON formats.
- The `--targetusername` option specifies the username for accessing the scratch org (the value in `SFDC_USERNAME`).

The `force:apex:test:run` command writes its test results in JUnit format. You can collect the results using industry-standard tools as shown in the following example.

```
stage('collect results') {
    junit keepLongStdio: true, testResults: 'tests/**/*-junit.xml'
}
```

Delete the Scratch Org

Salesforce reserves the right to delete a scratch org a specified number of days after it was created. You can also create a stage in your pipeline that uses `force:org:delete` to explicitly delete your scratch org when the tests complete. This cleanup ensures better management of your resources.

```
stage('Delete Test Org') {

    timeout(time: 120, unit: 'SECONDS') {
        rc = sh returnStatus: true, script: "${toolbelt}/sfdx force:org:delete
--targetusername ${SFDC_USERNAME} --noprompt"
        if (rc != 0) {
            error 'org deletion request failed'
        }
    }
}
```

```
}  
}
```

SEE ALSO:

- [sfdx-dreamhouse Sample GitHub Repo](#)
- [Pipeline-as-code with Multibranch Workflows in Jenkins](#)
- [TAP: Test Anything Protocol](#)
- [Configure Your Environment for Jenkins](#)
- [Salesforce CLI Command Reference \(Beta\)](#)

Continuous Integration with Travis CI

Travis CI is a cloud-based continuous integration (CI) service for building and testing software projects hosted on GitHub.

Setting up Salesforce DX with Travis CI is easy. See the `sfdx-travisci` GitHub sample and the Salesforce DX Trailhead modules to get started.

SEE ALSO:

- [sfdx-travisci Sample GitHub Repo](#)
- [Travis CI](#)

CHAPTER 8 Troubleshoot Salesforce DX

In this chapter ...

- [CLI Version Information](#)
- [Run CLI Commands on macOS Sierra \(Version 10.12\)](#)
- [Error: No defaultdevhubusername org found](#)
- [Unable to Work After Failed Org Authorization](#)
- [Error: Lightning Experience-Enabled Custom Domain Is Unavailable](#)

This guide is a work in progress. Log in to the Salesforce Success Community and let us know if you find a solution that would help other users so that we can incorporate it.

SEE ALSO:

[Salesforce Success Community](#)

CLI Version Information

Use these commands to view version information about the Salesforce DX CLI.

```
sfdx --version           // CLI version
sfdx plugins            // SalesforceDX plugin version
sfdx force --version    // Salesforce API version used by the CLI
```

Run CLI Commands on macOS Sierra (Version 10.12)

Some users who upgrade to macOS Sierra can't execute CLI commands. This is a general problem and not isolated to Salesforce DX. To resolve the issue, reinstall your Xcode developer tools.

Execute this command in Terminal:

```
xcode-select --install
```

If you still can't execute CLI commands, download the **Command Line Tools (macOS sierra) for Xcode 8** package from the Apple Developer website.

SEE ALSO:

[Apple Developer Downloads](#)

[Stack Overflow: Command Line Tools bash \(git\) not working - macOS Sierra Final Release Candidate](#)

Error: No defaultdevhubusername org found

Let's say you successfully authorize a DevHub org using the `--setDefaultdevhubusername` parameter. The username associated with the org is your default Dev Hub username. You then successfully create a scratch org without using the `--targetdevhubusername` parameter.

But when you try to create a scratch org another time using the same CLI command, you get this error:

```
Unable to invoke command. name: NoOrgFound message: No defaultdevhubusername org found
```

What happened?

Answer: You are no longer in the directory where you ran the authorization command. The directory from which you use the `--setDefaultdevhubusername` parameter matters.

If you run the authorization command from the root of your project directory, the `defaultdevhubusername` config value is set locally. The value applies only when you run the command from the same project directory. If you change to a different directory and run `force:org:create`, the local setting of the default Dev Hub org no longer applies and you get an error.

Solve the problem by doing one of the following.

- Set `defaultdevhubusername` globally so that you can run `force:org:create` from any directory.

```
sfdx force:config:set defaultdevhubusername=<devhubusername> --global
```

- Run `force:org:create` from the same project directory where you authorized your Dev Hub org.

- Use the `--targetdevhubusername` parameter with `force:org:create` to run it from any directory.

```
sfdx force:org:create --definitionfile <file> --targetdevhubusername <devhubusername>
--setalias my-scratch-org
```

- To check whether you've set configuration values globally or locally, use this command.

```
sfdx force:config:list
```

SEE ALSO:

[How Salesforce Developer Experience Changes the Way You Work \(Beta\)](#)

Unable to Work After Failed Org Authorization

Sometimes you try to authorize a Dev Hub or a scratch org using the Salesforce CLI or Force.com IDE 2, but you don't successfully log in to the org. The port remains open for the stray authorization process, and you can't use the CLI or IDE. To proceed, end the process manually.

macOS or Linux

To recover from a failed org authorization on macOS or Linux, use a terminal to kill the process running on port 1717.

1. From a terminal, run:

```
lsof -i tcp:1717
```


2. In the results, find the ID for the process that's using the port.
3. Run:

```
kill -9 <the process ID>
```

Windows

To recover from a failed org authorization on Windows, use the Task Manager to end the Node process.

1. Press **Ctrl+Alt+Delete**, then click **Task Manager**.
2. Select the **Process** tab.
3. Find the process named **Node**.

 **Note:** If you're a Node.js developer, you might have several running processes with this name.

4. Select the process that you want to end, and then click **End Process**.

Error: Lightning Experience-Enabled Custom Domain Is Unavailable

If you create a scratch org with `force:org:create`, and then immediately try to use it, you sometimes get an error after waiting a few minutes for the command to finish.

For example, if you try to open the new scratch org in a browser with `force:org:open`, you might get this error:

```
Waiting to resolve the Lightning Experience-enabled custom domain...  
ERROR running force:org:open: The Lightning Experience-enabled custom domain is unavailable.
```

The error occurs because it takes a few minutes for the Lightning Experience-enabled custom domain to internally resolve.

When using the CLI interactively, wait a few more minutes and run the command again. In a CI environment, however, you can avoid the error altogether by changing how long the CLI itself waits.


By default, the CLI waits 240 seconds (4 minutes) for the custom domain to become available. You can configure the CLI to wait longer by setting the `SFDX_DOMAIN_RETRY` environment variable to the number of seconds you want it to wait. For example, to wait 5 minutes (300 seconds):

```
export SFDX_DOMAIN_RETRY=300
```

If you want the CLI to bypass the custom domain check entirely, set `SFDX_DOMAIN_RETRY` to 0.

CHAPTER 9 Considerations for Salesforce DX (Beta)

Here are some known issues you might run into while using Salesforce DX.

 **Note:** This release contains a beta version of Salesforce DX, which means it's a high-quality feature with known limitations. Salesforce DX isn't generally available unless or until Salesforce announces its general availability in documentation or in press releases or public statements. We can't guarantee general availability within any particular time frame or at all. Make your purchase decisions only on the basis of generally available products and features. You can provide feedback and suggestions for Salesforce DX in the [Salesforce DX Beta](#) group in the Success Community.

Salesforce DX CLI

Limited Support for Shell Environments on Windows

Description: Salesforce CLI is tested on the Command Prompt (`cmd.exe`), Powershell, and Windows Subsystem for Linux (WSL) for Ubuntu. There are known issues in the cygwin and Min-GW environments. These environments might be tested and supported in a future release. For now, use a supported shell instead.

Workaround: None.

The `force:apex:test:run` Command Doesn't Finish Executing

Description: In certain situations, the `force:apex:test:run` command doesn't finish executing. Examples of these situations include a compile error in the Apex test or an Apex test triggering a pre-compile when another is in progress.

Workaround: Stop the command execution by typing control-C. If the command is part of a continuous integration (CI) job, try setting the environment variable `SFDX_PRECOMPILE_DISABLE=true`.

Dev Hub and Scratch Orgs

Default Views of `ScratchOrgInfo`, `ActiveScratchOrg`, and `NamespaceRegistry` are "Recently Viewed" Instead of "All"

Description: When you click the tabs, you see only items that you recently viewed which gives the impression that data is missing when it's not. This issue applies only to Lightning Experience. It does not affect the CLI.

Workaround: Create a view that lists all the items.

Person Accounts Don't Work in Developer Edition Scratch Orgs

Description: Person accounts work only in Enterprise Edition and Professional Edition scratch orgs.

Workaround: When working with person accounts, don't use a DE scratch org.

Source Management

`force:mdapi:convert` Ignores the `-d` Parameter

Description: If you run `force:mdapi:convert` with the `-d` parameter, the command correctly creates the specified directory if it doesn't exist. But the command then incorrectly outputs the converted source into the default directory, as defined by the `project-workspace.json` file.

Workaround: None.

`force:source:status` and `force:source:pull` Do Not Work Correctly After Deactivating a Flow

Description: Let's say you deactivate a Flow version by changing its state using the scratch org's Setup. The next time you run `force:source:status` or `force:source:pull`, the commands correctly recognize that the Flow version has changed. But the commands do not recognize that the FlowDefinition has changed, even though it has. As a result, the source status output or pulled source is incorrect.

Workaround: In Setup, edit the Flow's description and rerun the `force:source:status` or `force:source:pull` command.

Changes to Standard Fields on Standard Objects Not Being Pulled

Description: If you change a standard field of a standard object in the scratch org, then run `force:source:pull`, the field changes are not pulled to your project.

Workaround: None.

Pushing After Removing a Permission Set Locally Causes "Unknown" Error

Description: If you remove a permission set from your local project, then run the `force:source:push` command, you see the error message: Unknown.

Workaround: None.

`force:source:pull` Shows New Transformed Objects As Changed Instead of Added

Description: The CLI output for `force:source:pull` displays the state of transformed objects as Changed instead of Added.

Workaround: None.

Modified Search Layouts and Compact Layout Assignments on Standard Objects Are Not Tracked

Description: Let's say you modify a search layout or compact layout assignment on a standard object in your scratch org. When you next run the `force:source:pull` command, Salesforce DX does not track the search layout or compact layout assignment. As a result, you don't see the change in the output of the `force:source:pull` command or in your local project.

Workaround: If you make some other change to the standard object, the search layouts and compact layout assignments are tracked and pulled to the Salesforce DX project.

Unchanged Standard and Custom Fields Are Pulled to the Project

Description: The `force:source:pull` command is including unchanged standard and custom fields.

Workaround: None.

Lightning Components Not Pulled to Correct Directory

Description: Lightning components are not being pulled to the correct directory if they are pushed from a directory other than `main/default/aura/ComponentDir`.

Workaround: Place Lightning components in a folder under `main/default/aura/`.

QuickAction CustomObjectTranslations are not pulled if the original quickAction is not included in the package.xml

Description: The custom object translation that was pulled should include the new translation for the quick action.

Workaround: Wait until you have finished creating the quickAction and its customObjectTranslation before doing a `force:source:pull`.

Unable to Push Lookup Filters to a Scratch Org

Description: When you execute the `force:source:push` command to push the source of a relationship field that has a lookup filter, you sometimes get the following error:

```
duplicate value found: <unknown> duplicates value on record with  
id: <unknown> at line num, col num.
```

Workaround: None.

Package Development

New Terminology in CLI for Managed Package Password

Description: When you use the CLI to add an installation key to a package version or to install a key-protected package version, the parameter name of the key is `--installationkey`. When you view a managed package version in the Salesforce user interface, the same package attribute is called "Password".

Workaround: None. The Password field name in the user interface will be changed in a future release.