bitwarden | Help Center Article

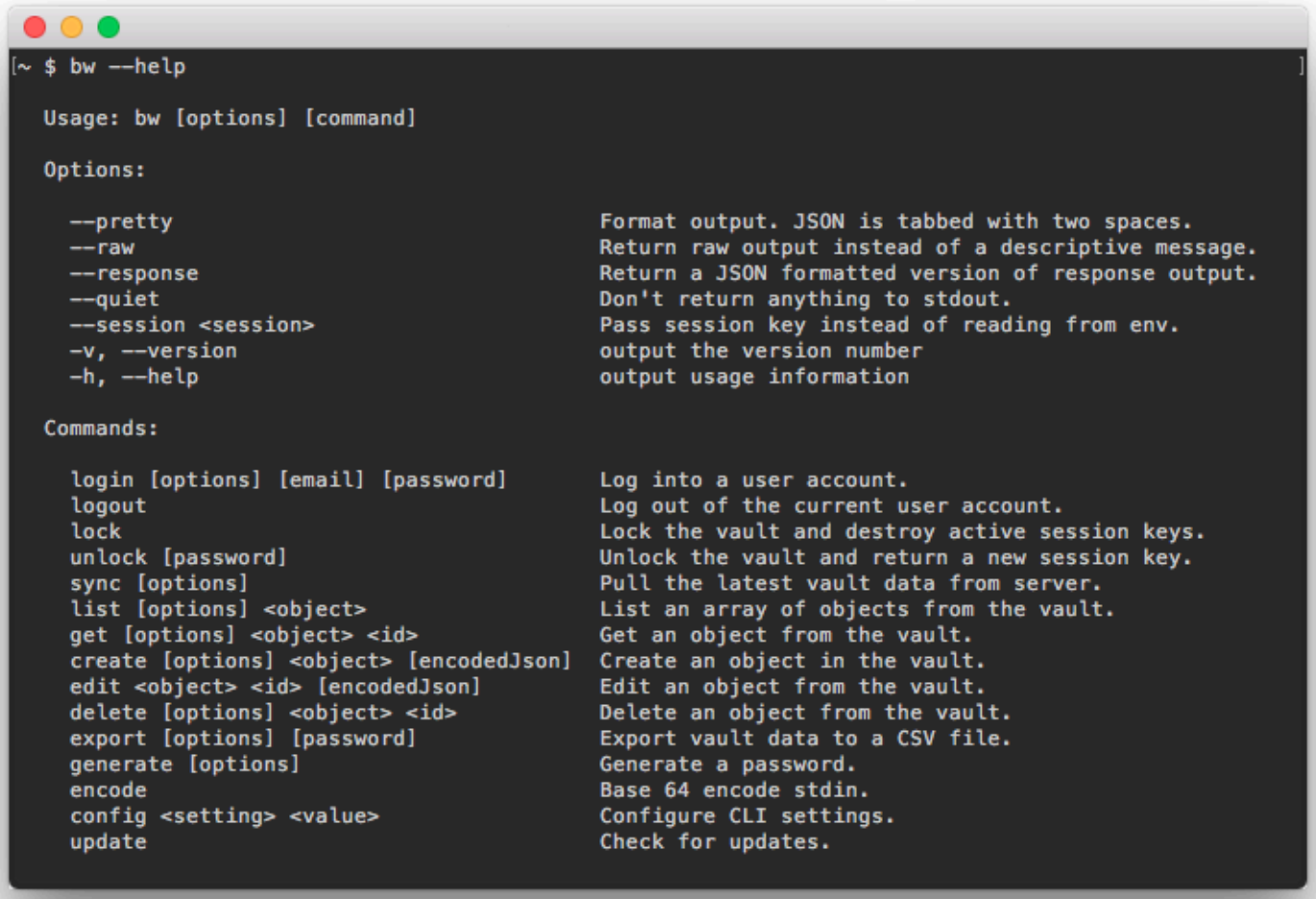
PASSWORD MANAGER > DEVELOPER TOOLS > CLI

# Password Manager CLI

View in the help center:
https://bitwarden.com/help/cli/

# Password Manager CLI

The Bitwarden command-line interface (CLI) is a powerful, fully-featured tool for accessing and managing your vault. Most features that you find in other Bitwarden client applications (desktop, browser extension, etc.) are available from the CLI.




Bitwarden CLI

The Bitwarden CLI is self-documented. From the command line, learn about the available commands using:

```Bash
bw --help
```

Or, pass `--help` as an option on any `bw` command to see available options and examples:

```
Bash

bw list --help

bw move --help
```

Most information you'll need can be accessed using `--help`, however this article replicates all that information and goes into greater depth on some topics.

## Download and install

The CLI can be used cross-platform on Windows, macOS, and Linux distributions. To download and install the Bitwarden CLI:

> **Note**
> For arm64 devices, install the CLI using `npm`.

### ⇒Native Executable

Natively packaged versions of the CLI are available for each platform and have no dependencies. Download using one of these links:

- Windows x64
- macOS x64
- Linux x64

Note that, when using the downloaded native executable, you'll need to add the executable to your PATH or else run commands from the directory the file is downloaded to.

> **Tip**
> In Linux and UNIX systems, you might get a `Permission denied` message. If you do, grant permission by running:

```
Bash

chmod +x </path/to/executable>
```

For each bundle of the Password Manager CLI available on GitHub, there is an OSS (e.g. `bw-oss-windows-2024.12.0.zip`) and non-OSS build (e.g. `bw-windows-2024.12.0.zip`). The non-OSS version is the default package distributed on distribution platforms and includes features under a non-OSS license, such as device approval commands, that the OSS version lacks.

> **Tip**
>
> The Bitwarden Password Manager CLI build pipeline creates SHA-256 checksum files that are available on GitHub. Learn how to validate checksums for the CLI.

### ⇒NPM

If you have Node.js installed on your system, you can install the CLI using NPM. Installing with NPM is the simplest way to keep your installation up-to-date and should be the **preferred method for those already comfortable with NPM**:

```
Bash

npm install -g @bitwarden/cli
```

View the package on npmjs.org.

> **Note**
>
> Installing the Bitwarden CLI on Linux systems using `npm` may require the build-essential dependency (or distribution equivalent) to be installed first. For example:
>
> ```
> Plain Text
>
> apt install build-essential
> ```

### ⇒Chocolatey

To install with Chocolatey:

```
Bash

choco install bitwarden-cli
```

View the package on community.chocolatey.org.

### ⇒Snap

To install with snap:

```
Bash

sudo snap install bw
```

View the package on snapcraft.io.

### ⇒Flatpak

The Bitwarden CLI is included with the Flatpak desktop app download. Install the Flatpak:

```
Bash

flatpak install flathub com.bitwarden.desktop
```

View the package on Flathub.

Run CLI commands using the following:

```
Bash

flatpak run --command=bw com.bitwarden.desktop <command>

# use a shell alias to authorize a session
alias bw "flatpak run --command=bw com.bitwarden.desktop"

bw <command>
```

## Log in

Before logging in, make sure your CLI is connected to the correct server (for example, EU cloud or self-hosted) using the config command (learn more). There are three methods for logging in to the Bitwarden CLI using the `login` command, each of which is suited to different situations. Please review the following options to determine which method to use:

- Using email and master password
- Using an API key
- Using SSO

> **Tip**
>
> No matter which option you use, a master password will be required to `unlock` the client in order to access data with a session key. The email and master password option will authenticate your identity and generate a session key simultaneously, however the API key or SSO will require you subsequent use of the `unlock` command to generate a session key if you will be working with data directly.
>
> **Users who don't have master passwords**, for example as a result of joining an organization using trusted devices, will not be able to access data using the CLI. There are, however, a few commands that do not require decrypted data and therefore can be used without a master password, including `config`, `encode`, `generate`, `update`, and `status`.

## Using email and password

Logging in with email and password is **recommended for interactive sessions**. To log in with email and password:

```
Bash

bw login
```

This will initiate a prompt for your **Email Address**, **Master Password**, and (if enabled) at **Two-step Login code**. The CLI currently supports two-step login via authenticator, email, or Yubikey.

You can string these factors together into a single command as in the following example, however this isn't recommended for security reasons:

```
Bash

bw login [email] [password] --method <method> --code <code>
```

See Enums for two-step login `<method>` values.

> **Tip**
>
> Getting prompted for additional authentication or getting a `Your authentication request appears to be coming from a bot.` error? Use your API Key `client_secret` to answer the authentication challenge. Learn more.

## Using an API key

Logging in with the personal API key is recommended for automated workflows, for providing access to an external application, or if your account uses a 2FA method not supported by the CLI (FIDO2 or Duo). To log in with the API key:

```
Bash

bw login --apikey
```

This will initiate a prompt for your personal `client_id` and `client_secret`. Once your session is authenticated using these values, you can use the `unlock` command. Learn more.

> **Tip**
>
> If your organization requires SSO, you can still use `--apikey` to log in to the CLI.

### Using API key environment variables

In scenarios where automated work is being done with the Bitwarden CLI, you can save environment variables to prevent the need for manual intervention at authentication.

| Environment variable name | Required value |
| --- | --- |
| BW_CLIENTID | `client_id` |
| BW_CLIENTSECRET | `client_secret` |

## Using SSO

Logging in with SSO is recommended if an organization requires SSO authentication. To log in with SSO:

```
Bash

bw login --sso
```

This will initiate the SSO authentication flow in your web browser. Once your session is authenticated, you can use the `unlock` command. Learn more.

> **Tip**
>
> If your organization requires SSO, you may alternatively use `--apikey` to log in to the CLI.

## Log in to multiple accounts

Like using account switching on other Bitwarden apps, the CLI has the ability to log in to multiple accounts simultaneously using the `BITWARDENCLI_APPDATA_DIR` environment variable pointing to the location of a `bw` configuration file, usually named `data.json`. You can, for example, set aliases in a `.bashrc` profile for two separate configurations:

```
Bash

alias bw-personal="BITWARDENCLI_APPDATA_DIR=~/.config/Bitwarden\ CLI\ Personal /path/to/bw $@"
alias bw-work="BITWARDENCLI_APPDATA_DIR=~/.config/Bitwarden\ CLI\ Work /path/to/bw $@"
```

Using this example, you could then use login to two accounts by running first `source /path/to/.bashrc`, followed by `bw-personal login` and `bw-work login`.

## Unlock

Using an API key or SSO to log in will require you to follow-up the `login` command with an explicit `bw unlock` if you'll be working with vault data directly.

Unlocking your vault generates a **session key** which acts as a decryption key used to interact with data in your vault. The session key must be used to perform any command that touches vault data (for example, `list`, `get`, `edit`). Session keys are valid until invalidated using `bw lock` or `bw logout`, however they will not persist if you open a new terminal window. Generate a new session key at any time using:

```
Bash

bw unlock
```

When you're finished, always end your session using the `bw lock` command.

## Unlock options

You can use the `--passwordenv <passwordenv>` or `--passwordfile <passwordfile>` options with `bw unlock` to retrieve your master password rather than enter it manually, for example:

1. The following will look for an environment variable `BW_PASSWORD`. If `BW_PASSWORD` is non-empty and has correct values, the CLI will successfully unlock and return a session key:

```
Bash

bw unlock --passwordenv BW_PASSWORD
```

2. The following will look for the file `~Users/Me/Documents/mp.txt` (which must have your master password as the first line). If the file is non-empty and has a correct value, the CLI will successfully unlock and return a session key:

```
Bash

bw unlock --passwordfile ~/Users/Me/Documents/mp.txt
```

> **⚠ Warning**
>
> If you use the `--passwordfile` option, protect your password file by locking access down to only the user who needs to run `bw unlock` and only providing read access to that user.

### Using a session key

When you unlock your vault using `bw login` with email and password or `bw unlock`, the CLI will return both an `export BW_SESSION` (Bash) and `env:BW_SESSION` (PowerShell) command, including your session key. Copy and paste the relevant entry to save the required environment variable.

With the BW_SESSION environment variable set, `bw` commands will reference that variable and can be run cleanly, for example:

```
Bash

export BW_SESSION="5PBYGU+5yt3RHcCjoeJKx/wByU34vokGRZjXpSH7Ylo8w=="

bw list items
```

Alternatively, if you don't set the environment variable, you can pass the session key as an option with each `bw` command:

```
Bash

bw list items --session "5PBYGU+5yt3RHcCjoeJKx/wByU34vokGRZjXpSH7Ylo8w=="
```

> **Tip**
>
> When you're finished, always end your session using the `bw lock` or `bw logout` commands. This will invalidate the active session key.

## Core Commands

### create

The `create` command creates a new object (`item`, `attachment`, and more) in your vault:

```
Bash

bw create (item|attachment|folder|org-collection) <encodedJson> [options]
```

The `create` command takes encoded JSON. A typical workflow for creating an object might look something like:

1. Use the `get template` command (see get core commands for details) to output the appropriate JSON template for the object type.
2. Use a command-line JSON processor like jq to manipulate the outputted template as required.
3. Use the `encode` command (see details) to encode the manipulated JSON.
4. Use the `create` command to create an object from the encoded JSON.

For example:

```
Bash

bw get template folder | jq '.name="My First Folder"' | bw encode | bw create folder
```

or

```
Bash

bw get template item | jq ".name=\"My Login Item\" | .login=$(bw get template item.login | jq '.use
rname="jdoe" | .password="myp@ssword123"')" | bw encode | bw create item
```

Upon successful creation, the newly created object will be returned as JSON.

### create other item types

The create command defaults to creating a login item, but you can use a command-line JSON processor like jq to change a `.type=` attribute to create other item types:

| Name | Value |
| --- | --- |
| Login | `.type=1` |
| Secure note | `.type=2` |
| Card | `.type=3` |
| Identity | `.type=4` |

For example, the following command will create a secure note:

```
Bash

bw get template item | jq '.type = 2 | .secureNote.type = 0 | .notes = "Contents of my Secure Note." | .name = "My Secure Note"' | bw encode | bw create item
```

> **Note**
>
> Notice in the above example that Secure Notes require a sub-template (`.secureNote.type`). You can view item type sub-templates using `bw get template` (see here for details).

### create attachment

The `create attachment` command attaches a file to an **existing** item.

Unlike other `create` operations, you don't need to use a JSON processor or `encode` to create an attachment. Instead, use the `--file` option to specify the file to attach and the `--itemid` option to specify the item to attach it to. For example:

```
Bash

bw create attachment --file ./path/to/file --itemid 16b15b89-65b3-4639-ad2a-95052a6d8f66
```

> **Tip**
>
> If you don't know the exact `itemid` you want to use, use `bw get item <search-term>` to return the item (see details), including its `id`.

### get

The `get` command retrieves a single object (`item`, `username`, `password`, and more) from your vault:

```
Bash

bw get (item|username|password|uri|totp|exposed|attachment|folder|collection|organization|org-collection|template|fingerprint) <id> [options]
```

The `get` command takes an item `id` or string for its argument. If you use a string (for example, anything other than an exact `id`), `get` will search your vault objects for one with a value that matches. For example, the following command would return a Github password:

```
Bash

bw get password Github
```

> **Note**
>
> The `get` command can **only return one result**, so you should use specific search terms. If multiple results are found, the CLI will return an error.

### get attachment

The `get attachment` command downloads a file attachment:

```
Bash

bw get attachment <filename> --itemid <id>
```

The `get attachment` command takes a `filename` and **exact** `id`. By default, `get attachment` will download the attachment to the current working directory. You can use the `--output` option to specify a different output directory, for example:

```
Bash

bw get attachment photo.png --itemid 99ee88d2-6046-4ea7-92c2-acac464b1412 --output /Users/myaccoun
t/Pictures/
```

> **Note**
>
> When using `--output`, the path **must** end a forward slash (`/`) to specify a directory or a filename (`/Users/myaccount/Pictures/photo.png`).

### get notes

The `get notes` command retrieves the note for any vault item:

```
Bash

bw get notes <id>
```

The `get notes` command takes an exact item `id` or string. If you use a string (for example, anything other than an exact `id`), `get notes` will search your vault objects for one with a value that matches. For example, the following command would return a Github note:

```
Bash

bw get notes Github
```

### get template

The `get template` command returns the expected JSON formatting for an object (`item`, `item.field`, `item.login`, and more):

```
Bash

bw get template (item|item.field|item.login|item.login.uri|item.card|item.identity|item.securenote|
folder|collection|item-collections|org-collection)
```

While you can use `get template` to output the format to your screen, the most common use-case is to pipe the output into a `bw create` operation, using a command-line JSON processor like jq and `bw encode` to manipulate the values retrieved from the template, for example:

```
Bash

bw get template folder | jq '.name="My First Folder"' | bw encode | bw create folder
```

> **Note**
>
> Any `item.xxx` template should be used as a sub-object to an `item` template, for example:
>
> ```
> Bash
>
> bw get template item | jq ".name=\"My Login Item\" | .login=$(bw get template item.login | j
> q '.username="jdoe" | .password="myp@ssword123"')" | bw encode | bw create item
> ```

### get fingerprint

Retrieve the `fingerprint` phrase of a user. You may specify `userId` directly, or use the shortcut `me` to get your own fingerprint phrase:

```
Plain Text

bw get fingerprint <userId>
```

```
Plain Text

bw get fingerprint me
```

### edit

The `edit` command edits an object (`item`, `item-collections`, etc.) in your vault:

```
Bash

bw edit (item|item-collections|folder|org-collection) <id> [encodedJson] [options]
```

The `edit` command takes an **exact** `id` (the object to edit) and encoded JSON (edits to be made). A typical workflow might look something like:

1. Use the `get` command (see details) to output the object to edit.
2. Use a command-line JSON processor like jq to manipulate the outputted object as required.
3. Use the `encode` command (see details) to encode the manipulated JSON.
4. Use the `edit` command (including the object `id`) to edit the object.

For example, to edit the password of a login item:

```
Bash

bw get item 7ac9cae8-5067-4faf-b6ab-acfd00e2c328 | jq '.login.password="newp@ssw0rd"' | bw encode |
bw edit item 7ac9cae8-5067-4faf-b6ab-acfd00e2c328
```

Or, to edit the collection(s) an item is in:

```
Bash

echo '["5c926f4f-de9c-449b-8d5f-aec1011c48f6"]' | bw encode | bw edit item-collections 28399a57-73a
0-45a3-80f8-aec1011c48f6 --organizationid 4016326f-98b6-42ff-b9fc-ac63014988f5
```

Or, to edit a collection:

```
Bash

bw get org-collection ee9f9dc2-ec29-4b7f-9afb-aac8010631a1 --organizationid 4016326f-98b6-42ff-b9fc
-ac63014988f5 | jq '.name="My Collection"' | bw encode | bw edit org-collection ee9f9dc2-ec29-4b7f-
9afb-aac8010631a1 --organizationid 4016326f-98b6-42ff-b9fc-ac63014988f5
```

The `edit` command will perform a **replace** operation on the object. Once completed, the updated object will be returned as JSON.

### list

The `list` command retrieves an array of objects (`items`, `folders`, `collections`, and more) from your vault:

```
Bash

bw list (items|folders|collections|organizations|org-collections|org-members) [options]
```

Options for the `list` command are **filters** used to dictate what will be returned, including `--url <url>`, `--folderid <folderid>`, `--collectionid <collectionid>`, `--organizationid <organizationid>` and `--trash`. Any filter will accept `null` or `notnull`. Combining multiple filters in one command will perform an OR operation, for example:

```
Bash

bw list items --folderid null --collectionid null
```

This command will return items that aren't in a folder or collection.

Additionally, you can **search** for specific objects using `--search <search-term>`. Combining filter and search in one command will perform an AND operation, for example:

```
Bash

bw list items --search github --folderid 9742101e-68b8-4a07-b5b1-9578b5f88e6f
```

This command will search for items with the string `github` in the specified folder.

### delete

The `delete` command deletes an object from your vault. `delete` takes **only an exact** `id` for its argument.

```
Bash

bw delete (item|attachment|folder|org-collection) <id> [options]
```

By default, `delete` will send an item to the Trash, where it will remain for 30 days. You can permanently delete an item using the `-p, --permanent` option.

```
Bash

bw delete item 7063feab-4b10-472e-b64c-785e2b870b92 --permanent
```

To delete an `org-collection`, you'll also need to specify `--organizationid <organizationid>`. See Organization IDs.

> **⚠ Warning**
>
> While items that are deleted using `delete` can be recovered using the `restore` command for up to 30 days (see details), items that are deleted using `delete --permanent` **are completely removed and irrecoverable.**

### restore

The `restore` command restores a deleted object from your trash. `restore` takes **only an exact** `id` for its argument.

```
Bash

bw restore (item) <id> [options]
```

For example:

```
Bash

bw restore item 7063feab-4b10-472e-b64c-785e2b870b92
```

### send

The `send` command creates a Bitwarden Send object for ephemeral sharing. This section will detail simple `send` operations, however send is a highly flexible tool and we recommend referring to the dedicated article on Send from CLI.

To create a simple text Send:

```
Bash

bw send -n "My First Send" -d 7 --hidden "The contents of my first text Send."
```

To create a simple file Send:

```
Bash

bw send -n "A Sensitive File" -d 14 -f /Users/my_account/Documents/sensitive_file.pdf
```

### receive

The `receive` command accesses a Bitwarden Send object. To receive a Send object:

```
Bash

bw receive --password passwordforaccess https://vault.bitwarden.com/#/send/yawoill8rk6VM6zCATXv2A/9WN8wD-hzsDJjfnXLeNc2Q
```

## Organizations commands

### Organization IDs

Accessing an organization from the CLI requires knowledge of an ID for your organization, as well as IDs for individual members and collections.

Retrieve this information directly from the CLI using `bw list`, for example:

```
Bash

bw list organizations
bw list org-members --organizationid 4016326f-98b6-42ff-b9fc-ac63014988f5
bw list org-collections --organizationid 4016326f-98b6-42ff-b9fc-ac63014988f5
```

**Tip**

You can `bw list` both `collections` and `org-collections`. The `bw list collections` command will list all collections, agnostic of which organization they belong to. `bw list org-collections` will list only collections that belong to the organization specified using `--organizationid`.

### move

> **ⓘ Note**
>
> **August 2021**: The `share` command has been changed to `move`. Find out more.

The `move` command transfers a vault item to an organization:

```
Bash

bw move <itemid> <organizationid> [encodedJson]
```

The `move` command requires you to `encode` a collection ID, and takes an **exact** `id` (the object to share) and an **exact** `organizationid` (the organization to share the object to). For example:

```
Bash

echo '["bq209461-4129-4b8d-b760-acd401474va2"]' | bw encode | bw move ed42f44c-f81f-48de-a123-ad010
13132ca dfghbc921-04eb-43a7-84b1-ac74013bqb2e
```

Once completed, the updated item will be returned.

### confirm

The `confirm` command confirms invited members to your organization who have accepted their invitation:

> **⚠ Warning**
>
> Before administering the `confirm` command, it is strongly advised that administrators validate the legitimacy of a request by ensuring that the fingerprint phrase self-reported by the user matches the fingerprint phrase associated with the user you expect to be confirmed:
>
> - From the Admin Console, you can view a user's associated fingerprint phrase during the confirm step.
> - From the CLI, you can view a user's associated fingerprint phrase with the command `bw get fingerprint <id>`, where `<id>` is that member's user identifier. User identifiers can be retrieved with the Public API.
>
> Once a user is confirmed, they have the ability to decrypt organization data, so ensuring users' self-reported fingerprint phrases match expected values is an important step prior to confirming.

```
Bash

bw confirm org-member <id> --organizationid <orgid>
```

The `confirm` command takes an **exact** member `id` and an **exact** `organizationID`, for example:

```
Bash

bw confirm org-member 7063feab-4b10-472e-b64c-785e2b870b92 --organizationid 310d5ffd-e9a2-4451-af87
-ea054dce0f78
```

### Device approval

Allows admins and owners to manage device approval requests where a user has requested admin approval.

> **ⓘ Note**
>
> At this time, bulk device approval is only available for the Bitwarden CLI client downloaded from Bitwarden.com.

> **⚠ Warning**
>
> In most scenarios, users are able to approve their own login requests, and admin device approval is not necessary. See Add a trusted device. Automatic or bulk approval of admin device approval requests neglect verification steps that administrators can perform in order to ensure a request is legitimate, such as checking the user's reported Fingerprint Phrase.
>
> Bitwarden recommends that significant security controls such as IdP credential standards, IdP MFA, and IdP device registration and trust be reviewed before enabling and using bulk device approval.

The `list` command will show all pending device approval requests for an organization:

```
Plain Text

bw device-approval list --organizationid <organization_Id>
```

The `approve` command is used to approve pending device authorization requests for an organization:

```
Plain Text

bw device-approval approve --organizationid <organizationId> <requestId>
```

Similarly, `approve-all` command can be used to approve all current pending requests:

```
Plain Text

bw device-approval approve-all --organization <organizationId>
```

To `deny` a pending authorization request:

```
Plain Text

bw device-approval deny --organizationid <organizationId> <requestId>
```

To `deny-all` pending authorization requests:

```
Plain Text

bw device-approval deny-all --organizationid <organizationId>
```

## Other commands

### config

The `config` command specifies settings for the Bitwarden CLI to use:

```
Bash

bw config server <setting> [value]
```

A primary use of `bw config` is to connect your CLI to a self-hosted Bitwarden server:

```
Bash

bw config server https://your.bw.domain.com
```

**Tip**

Connect to the Bitwarden EU server by running the following command:

```
Bash

bw config server https://vault.bitwarden.eu
```

Pass `bw config server` without a value to read the server you're connected to.

Users with unique setups may elect to specify the URL of each service independently. Note that any subsequent use of the config command will overwrite all previous specifications, so this must be run as a single command each time you make a change:

```
Bash

bw config server --web-vault <url> \
  --api <url> \
  --identity <url> \
  --icons <url> \
  --notifications <url> \
  --events <url> \
  --key-connector <url>
```

**Note**

The `bw config server --key-connector <url>` command is required if your organization uses Key Connector and you're using the `--apikey` option to login after having removed your master password.

Contact an organization owner to get the required URL.

### sync

The `sync` command downloads your encrypted vault from the Bitwarden server. This command is most useful when you have changed something in your Bitwarden vault on another client application (for example web vault, browser extension, mobile app) since logging in on the CLI.

```
Bash

bw sync
```

You can pass the `--last` option to return only the timestamp (ISO 8601) of the last time a sync was performed.

> **Tip**
>
> It's important to know that `sync` **only performs a pull** from the server. Data is automatically pushed to the server any time you make a change to your vault (for example, `create`, `edit`, `delete`).

### encode

The `encode` command Base 64 encodes stdin. This command is typically used in combination with a command-line JSON processor like jq when performing `create` and `edit` operations, for example:

```
Bash

bw get template folder | jq '.name="My First Folder"' | bw encode | bw create folder

bw get item 7ac9cae8-5067-4faf-b6ab-acfd00e2c328 | jq '.login.password="newp@ssw0rd"' | bw encode |
bw edit item 7ac9cae8-5067-4faf-b6ab-acfd00e2c328
```

### import

The `import` command imports data from a Bitwarden export or other supported password management application. The command must be pointed to a file and include the following arguments:

```
Bash

bw import <format> <path>
```

For example:

```
Bash

bw import lastpasscsv /Users/myaccount/Documents/mydata.csv
```

> **Tip**
>
> Bitwarden supports lots of formats for import, too many to list here! Use `bw import --formats` to return the list in your CLI, or see here.
>
> If you are importing an encrypted .json file that you've created with a password, you will be prompted to enter the password before import completes.

### import to an organization vault

Using the import command with the –organizationid option, you can import data to an organization vault:

```
Plain Text

bw import --organizationid cf14adc3-aca5-4573-890a-f6fa231436d9 bitwardencsv ./from/source.csv
```

## export

The `export` command exports vault data as a `.json` or `.csv`, encrypted .json, or as a `.zip` with attachments:

```
Bash

bw export [--output <filePath>] [--format <format>] [--password <password>] [--organizationid <orgid>]
```

By default, the `export` command will generate a `.csv` (equivalent to specifying `--format csv`) to the current working directory, however you can specify:

- `--format json` to export a `.json` file
- `--format encrypted_json` to export an encrypted .json file
  - `--password <password>` to specify a password to use to encrypt `encrypted_json` exports instead of your account encryption key
- `--format zip` to export a `.zip` that includes your attachments
- `--output <path>` to export to a specific location
- `--raw` to return the export to stdout instead of to a file

### export from an organization vault

Using the `export` command with the `--organizationid` option, you can export an organization vault:

```
Bash

bw export --organizationid 7063feab-4b10-472e-b64c-785e2b870b92 --format json --output /Users/myacc
ount/Downloads/
```

### generate

The `generate` command generates a strong password or passphrase:

```
Bash

bw generate [--lowercase --uppercase --number --special --length <length> --passphrase --separator
<separator> --words <words>]
```

By default, the `generate` command will generate a 14-character password with uppercase characters, lowercase characters, and numbers. This is the equivalent of passing:

```
Bash

bw generate -uln --length 14
```

You can generate more complex passwords using the options available to the command, including:

- `--uppercase`, `-u` (include uppercase)
- `--lowercase`, `-l` (include lowercase)
- `--number`, `-n` (include numbers)
- `--special`, `-s` (include special characters)
- `--length <length>` (length of the password, min of 5)

### generate a passphrase

Using the `generate` command with the `--passphrase` option, you can generate a passphrase instead of a password:

```
Bash

bw generate --passphrase --words <words> --separator <separator>
```

By default, `bw generate --passphrase` will generate a three-word passphrase separated by a dash (`-`). This is the equivalent of passing:

```
Bash

bw generate --passphrase --words 3 --separator -
```

You can generate a complex passphrase using the options available to the command, including:

- `--words <words>` (number of words)
- `--separator <separator>` (separator character)
- `--capitalize`, `-c` (include to title-case the passphrase)
- `--includeNumber` (Include a single numerical character in your passphrase)

### update

The `update` command checks whether your Bitwarden CLI is running the most recent version. `update` **doesn't automatically update the CLI for you.**

```
Bash

bw update
```

If a new version is detected, you'll need to download the new version of the CLI using the printed URL for the executable, or using the tools available for the package manager you used to download the CLI (for example, `npm install -g @bitwarden/cli`).

### status

The `status` command returns status information about the Bitwarden CLI, including configured server URL, timestamp for the last sync (ISO 8601), user email and ID, and the vault status.

```
Bash

bw status
```

Status will return information as a JSON object, for example:

```Bash
{
  "serverUrl": "https://bitwarden.example.com",
  "lastSync": "2020-06-16T06:33:51.419Z",
  "userEmail": "user@example.com",
  "userId": "00000000-0000-0000-0000-000000000000",
  "status": "unlocked"
}
```

`status` may be one of the following:

- `"unlocked"`, indicating you are logged in and your vault is unlocked (a `BW_SESSION` key environment variable is saved with an active session key)
- `"locked"`, indicating you are logged in but your vault is locked (**no** `BW_SESSION` key environment variable is saved with an active session key)
- `"unauthenticated"`, indicating you aren't logged in

> **Tip**
>
> When `"status": "unauthenticated"`, `lastSync`, `userEmail`, and `userID` will always return `null`.

### serve

The `serve` command starts a local express web server that can be used to take all actions accessible from the CLI in the form of RESTful API calls from an HTTP interface.

```Bash
bw serve --port <port> --hostname <hostname>
```

By default, `serve` will start the web server at port 8087 however you can specify an alternate port with the `--port` option.

By default, `serve` will bind your API web server to `localhost` however you can specify an alternate hostname with the `--hostname` option. API requests can only be made from the bound hostname.

By default, `serve` will block any request with an `Origin` header. You can circumvent this protection using the `--disable-origin-protection` option, however **this is not recommended**.

> **⚠ Warning**
> You can specify `--hostname all` for no hostname binding, however this will allow any machine on the network to make API requests.

View the API spec for help making calls with `serve`.

### Debug

The debug environment variable can be added for additional troubleshooting information.

```
Plain Text

export BITWARDENCLI_DEBUG=true
```

## Appendices

### Global options

The following options are available globally:

| Option | Description |
| --- | --- |
| `--pretty` | Format output. JSON is tabbed with two spaces. |
| `--raw` | Return raw output instead of a descriptive message. |
| `--response` | Return a JSON formatted version of response output. |
| `--quiet` | Don't return anything to stdout. You might use this option, for example, when piping a credential value to a file or application. |
| `--nointeraction` | Do not prompt for interactive user input. |

| Option | Description |
| --- | --- |
| `--session <session>` | Pass session key instead of reading from an environment variable. |
| `-v, --version` | Output the Bitwarden CLI version number. |
| `-h, --help` | Display help text for the command. |

## ZSH shell completion

The Bitwarden CLI includes support for ZSH shell completion. To setup shell completion, use one of the following methods:

1. **Vanilla ZSH:** Add the following line to your `.zshrc` file:

```
Bash

eval "$(bw completion --shell zsh); compdef _bw bw;"
```

2. **Vanilla (vendor-completions):** Run the following command:

```
Bash

bw completion --shell zsh | sudo tee /usr/share/zsh/vendor-completions/_bw
```

3. zinit: Run the following commands:

```
Bash

bw completion --shell zsh > ~/.local/share/zsh/completions/_bw
zinit creinstall ~/.local/share/zsh/completions
```

## Using self-signed certificates

If your self-hosted Bitwarden server exposes a self-signed TLS certificate, specify the Node.js environment variable NODE_EXTRA_CA_CERTS:

Bash

```Bash
export NODE_EXTRA_CA_CERTS="absolute/path/to/your/certificates.pem"
```

PowerShell

```Bash
$env:NODE_EXTRA_CA_CERTS="absolute/path/to/your/certificates.pem"
```

## Enums

The following tables enumerate values required in documented scenarios:

### Two-step login methods

Used to specify which two-step login method to use when logging in:

| Name | Value |
| --- | --- |
| Authenticator | 0 |
| Email | 1 |
| YubiKey | 3 |

> **Note**
>
> FIDO2 and Duo are not supported by the CLI.

### Item types

Used with the `create` command to specify a vault item type:

| Name | Value |
| --- | --- |
| Login | 1 |
| Secure Note | 2 |
| Card | 3 |
| Identity | 4 |

### Login URI match types

Used with the `create` and `edit` command to specify URI match detection behavior for a login item:

| Name | Value |
| --- | --- |
| Domain | 0 |
| Host | 1 |
| Starts With | 2 |
| Exact | 3 |
| Regular Expression | 4 |
| Never | 5 |

### Field types

Used with the `create` and `edit` commands to configure custom fields:

| Name | Value |
| --- | --- |
| Text | 0 |
| Hidden | 1 |
| Boolean | 2 |

### Organization user types

Indicates a user's type:

| Name | Value |
| --- | --- |
| Owner | 0 |
| Admin | 1 |
| User | 2 |
| Manager | 3 |
| Custom | 4 |

### Organization user statuses

Indicates a user's status within the organization:

| Name | Value |
| --- | --- |
| Invited | 0 |
| Accepted | 1 |
| Confirmed | 2 |
| Revoked | -1 |