bitwarden | Help Center Article

SECRETS MANAGER › DEVELOPER TOOLS

Secrets Manager CLI

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

Secrets Manager CLI

The Secrets Manager command-line interface (CLI) is a powerful tool for retrieving and injecting your secrets. The Secrets Manager CLI can be used to organize your vault with create, delete, edit, and list your secrets and projects. The Secrets Manager CLI has two run options:

- Standard usage
- Docker usage

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

```bash
bws --help, -h
```

Download and install

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

Download the Secrets Manager CLI from https://github.com/bitwarden/sdk/releases.

> **Note**
>
> 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.

Run with Docker

The Secrets Manager CLI can also run with Docker. An example Dockerfile can be located in the Bitwarden Secrets Manager SDK repository.

You can run the Docker image with the following:

```
docker run --rm -it bitwarden/bws --help
```

> **Note**
>
> If you want to use identical config file paths on your host and in the container, the parent directory must exist on both.

Authentication

The Secrets Manager CLI can be logged in to using an access token generated for a particular machine account. This means that **only secrets and projects which the machine account has access to** may be interacted with using the CLI. There are a few ways you can authenticate a CLI session:

⇒Environment variable

You can authenticate a CLI session by saving an environment variable `BWS_ACCESS_TOKEN` with the value of your access token, for example:

```bash
export BWS_ACCESS_TOKEN=0.48c78342-1635-48a6-accd-afbe01336365.C0tMmQqHnAp1h0gL8bngprlPOYutt0:B3h5D
+YgLvFiQhWkIq6Bow==
```

⇒Inline

You can authenticate individual CLI requests using the `-t`, `--access-token` flag with any individual command, for example:

```bash
bws secret list --access-token 0.48c78342-1635-48a6-accd-afbe01336365.C0tMmQqHnAp1h0gL8bngprlPOYutt
0:B3h5D+YgLvFiQhWkIq6Bow==
```

⚠ **Warning**

If your workflow uses many separate sessions (where each use of an access token to authenticate constitutes a "session") to make requests from the same IP address in a short span of time, you may encounter rate limits.

Commands

Commands are used to interact with the Secrets Manager CLI. Secrets and Projects can be read or written to depending on the permissions given to your specific access token. For additional details regarding the commands available for `secret` and `project`, use for example:

- `bws run --help`
- `bws secret --help`
- `bws project --help`

ⓘ **Note**

As of the Secrets Manager version 0.3.0, CLI syntax has been changed. The command to list secrets, for example has changed from `bws list secrets` to `bws secret list`.

The old syntax will temporarily remain supported in the Secrets Manager CLI. If you are not sure what version of the Secrets Manager CLI you're using, enter `bws --version`.

run

The `run` command runs commands with secrets injected as environment variables, enabling you to easily adapt existing development projects and scripts to use secure secrets management.

⚠ Warning

Only execute commands that you trust. The `run` command executes the commands you specify in your shell, so you should not use it to execute binaries, shell scripts, or ad-hoc shell commands that you do not trust. Untrusted executables can contain command injections or other malicious behavior that would gain access to secrets when running inside of `bws run`.

⇒Single command

You can execute single commands using `bws run -- 'your-command'`:

```
Bash

# run an npm project with secrets injected
bws run -- 'npm run start'
```

⇒Multiple commands

Multiple shell commands can be executed by wrapping them in single-quotes. Wrapping multiple command in single-quotes will ensure that the entirety of your command is passed to the run command before your shell interprets special characters (such as `$`, `&`, `;`, `"`, etc.):

```
Plain Text

# start a container stack, execute a script, and tear down the container stack
bws run -- 'docker compose up -d && ./second-command.sh; docker compose down'

# echo a secret's value by name
bws run -- 'echo "$secret_name"'
```

Tip

Most command-line utilities are subject to the limitations of POSIX. POSIX-compliance requires that environment variable names only consist of alpha-numeric characters or underscores and may only begin with letters or an underscore.

The Secrets Manager CLI will still set secret names that are not POSIX-compliant as environment variables, however they may only be accessible from programs that are not limited by POSIX-compliance. Refer to the section describing the `--uuids-as-keynames` argument for an easy way to ensure POSIX-compliant environment variable names for your secrets.

run --project-id

Use the `--project-id` option with the `run` command to inject secrets from a single project, for example:

```
Bash

bws run --project-id 7b006643-89c1-4202-a5ca-90510f566030 -- echo "only secrets from the specified
project will be available"
```

run --shell

The `run` command defaults to using `sh` on Linux and macOS and PowerShell on Windows. Use the `--shell` option with the `run` command to run with another installed shell, for example:

```bash
bws run --shell fish -- echo “running a command with the Fish shell”
```

run --no-inherit-env

Use the `--no-inherit-env` option with the run command to execute your process without inheriting most of the environment variables from your shell, for example:

Tip

While the `--no-inherit-env` argument attempts to drop environment variables from your shell, it will always inherit `$PATH`. Additionally, some environment variables (`$PWD`, `$SHLVL`, etc.) will be set automatically by the shell itself, and therefore likely be present.

```bash
bws run --no-inherit-env -- echo "running a command with a minimal environment"
```

Warning

The `--no-inherit-env` argument is an easy way to drop environment variables from your shell that may conflict with the process being executed. This option **does not create a sandbox**. The process you execute will have the same access to your system that any other non-sandboxed applications would.

run --uuids-as-keynames

By default, the `run` command will take secret names and set those as environment variables in the process being executed. Use the `--uuids-as-keynames` argument with the `run` command to use POSIX-compliant secret IDs as environment variable names instead, for example:

```bash
# echo a secret’s value by its POSIX-compliant UUID
bws run --uuids-as-keynames -- 'echo $_64246aa4_70b3_4332_8587_8b1284ce6d76'
```

Alternatively, you can set `BWS_UUIDS_AS_KEYNAMES=true` as an environment variable to have the same effect as passing the argument.

Tip

Since UUIDS contain hyphens, and sometimes begin with numbers, the `--uuids-as-keynames` argument will replace hyphens with underscores and always prepend an underscore to secret UUIDS to ensure POSIX-compliance. For example, as secret with the ID `64246aa4-70b3-4332-8587-8b1284ce6d76` is converted to `_64246aa4_70b3_4332_8587_8b1284ce6d76`.

secret

The `secret` command is used to access, manipulate, and create secrets. As with all commands, secrets and projects outside your access token's scope of access cannot be read or written-to.

secret create

Use `bws secret create` to create a new secret. This command requires a `KEY`, `VALUE`, and `PROJECT_ID`:

Bash

```bash
bws secret create <KEY> <VALUE> <PROJECT_ID>
```

Optionally, you can add a note using the `--note <NOTE>` option. For example:

Bash

```bash
bws secret create SES_KEY 0.982492bc-7f37-4475-9e60 f588b2f2-4780-4a78-be2a-b02d014d622f --note "API Key for AWS SES"
```

This command, by default, will return a JSON object and save the secret to Secrets Manager. You can alter the output format using the `--output` flag (learn more).

Bash

```
{
  "object": "secret",
  "id": "be8e0ad8-d545-4017-a55a-b02f014d4158",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "projectId": "e325ea69-a3ab-4dff-836f-b02e013fe530",
  "key": "SES_KEY",
  "value": "0.982492bc-7f37-4475-9e60",
  "note": "API Key for AWS SES",
  "creationDate": "2023-06-28T20:13:20.643567Z",
  "revisionDate": "2023-06-28T20:13:20.643567Z"
}
```

secret delete

Use `bws secret delete` to delete one or more secrets designated by the `SECRET_IDS`.

```bash
bws secret delete <SECRET_IDS>
```

To delete a single secret with the `id be8e0ad8-d545-4017-a55a-b02f014d4158`:

```bash
bws secret delete be8e0ad8-d545-4017-a55a-b02f014d4158
```

For multiple secrets where the `ids` are `382580ab-1368-4e85-bfa3-b02e01400c9f` and `47201c5c-5653-4e14-9007-b02f015b2d82`:

```bash
bws secret delete 382580ab-1368-4e85-bfa3-b02e01400c9f 47201c5c-5653-4e14-9007-b02f015b2d82
```

Output:

```bash
1 secret deleted successfully.
```

secret edit

To edit a secret, the following structure will apply changes to the chosen value. From the CLI this commands can edit the secret `KEY`, `VALUE`, `NOTE`, or `PROJECT_ID`.

```bash
bws secret edit <SECRET_ID> --key <KEY> --value <VALUE> --note <NOTE> --project-id <PROJECT_ID>
```

For example, if you wish to add a note to an existing secret:

```bash
bws secret edit be8e0ad8-d545-4017-a55a-b02f014d4158 --note "I am adding a note"
```

> **Note**
>
> Include quotation marks around the string when editing a `NOTE` containing spaces.

To edit multiple fields where `SES_KEY2` is the new `key` and `0.1982492bc-7f37-4475-9e60` is the new `value`:

```bash
bws secret edit be8e0ad8-d545-4017-a55a-b02f014d4158 --key SES_KEY2 --value 0.1982492bc-7f37-4475-9
e60
```

Output:

```bash
{
  "object": "secret",
  "id": "be8e0ad8-d545-4017-a55a-b02f014d4158",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "projectId": "e325ea69-a3ab-4dff-836f-b02e013fe530",
  "key": "SES_KEY2",
  "value": "0.1982492bc-7f37-4475-9e60",
  "note": "I am adding a note",
  "creationDate": "2023-06-28T20:13:20.643567Z",
  "revisionDate": "2023-06-28T20:45:37.46232Z"
}
```

secret get

Use `bws secret get` to retrieve a specific secret:

```bash
bws secret get <SECRET_ID>
```

By default, this command will retrieve the secret object with the `SECRET_ID`.

```bash
bws secret get be8e0ad8-d545-4017-a55a-b02f014d4158
```

By default, `get` will return objects as a JSON array, as shown in the following example. You can alter the output format using the `--output` flag (learn more).

```bash
{
  "object": "secret",
  "id": "be8e0ad8-d545-4017-a55a-b02f014d4158",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "projectId": "e325ea69-a3ab-4dff-836f-b02e013fe530",
  "key": "SES_KEY",
  "value": "0.982492bc-7f37-4475-9e60",
  "note": "",
  "creationDate": "2023-06-28T20:13:20.643567Z",
  "revisionDate": "2023-06-28T20:13:20.643567Z"
}
```

secret list

To list the secrets the machine account can access, use the following command:

```bash
bws secret list
```

You can also list only the secrets in a specific project by using the following command, where `e325ea69-a3ab-4dff-836f-b02e013fe530` represents a project identifier:

```bash
bws secret list e325ea69-a3ab-4dff-836f-b02e013fe530
```

By default, `list` will return objects as a JSON array, as in the following example. You can alter the output format using the `--output` flag (learn more).

Bash

```
[
  {
    "object": "secret",
    "id": "382580ab-1368-4e85-bfa3-b02e01400c9f",
    "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
    "projectId": "e325ea69-a3ab-4dff-836f-b02e013fe530",
    "key": "Repository 1",
    "value": "1234567ertthrjytkuy",
    "note": "Main Repo",
    "creationDate": "2023-06-27T19:25:15.822004Z",
    "revisionDate": "2023-06-27T19:25:15.822004Z"
  },
  {
    "object": "secret",
    "id": "be8e0ad8-d545-4017-a55a-b02f014d4158",
    "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
    "projectId": "e325ea69-a3ab-4dff-836f-b02e013fe530",
    "key": "SES_KEY",
    "value": "0.982492bc-7f37-4475-9e60",
    "note": "",
    "creationDate": "2023-06-28T20:13:20.643567Z",
    "revisionDate": "2023-06-28T20:13:20.643567Z"
  }
]
```

project

The project command is used to access, manipulate, and create projects. The scope of access assigned to your machine account will determine what actions can be completed with the `project` command.

> **Note**
>
> Projects can be created by a machine account with read-only access. However, existing projects that were not created by the machine account cannot be edited without **read** and **write** access.

project create

Use `bws project create` to create a new project. This command requires a `NAME`.

Bash

```
bws project create <NAME>
```

In this example, a project will be created with the name `My project`.

Bash

```
bws project create "My project"
```

By default, `bws project create` will return objects as a JSON array, as in the following example. You can alter the output format using the `--output` flag (learn more).

Bash

```
{
  "object": "project",
  "id": "1c80965c-acb3-486e-ac24-b03000dc7318",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "name": "My project",
  "creationDate": "2023-06-29T13:22:37.942559Z",
  "revisionDate": "2023-06-29T13:22:37.942559Z"
}
```

project delete

Use `bws project delete` to delete one or more projects designated by the `PROJECT_IDS`.

Bash

```
bws project delete <PROJECT_IDS>
```

For a single project where `f1fe5978-0aa1-4bb0-949b-b03000e0402a` represents the `PROJECT_ID`:

Bash

```
bws project delete f1fe5978-0aa1-4bb0-949b-b03000e0402a
```

For multiple projects where `1c80965c-acb3-486e-ac24-b03000dc7318` and f`277fd80-1bd2-4532-94b2-b03000e00c6c` represent the `PROJECT_IDS`:

```bash
bws project delete 1c80965c-acb3-486e-ac24-b03000dc7318 f277fd80-1bd2-4532-94b2-b03000e00c6c
```

Output:

```bash
1 project deleted successfully.
```

project edit

Using the `edit` command you can change the name of a project with the following input:

```bash
bws project edit <PROJECT_ID> --name <NEW_NAME>
```

For example, this command will change the project name to `My project 2`.

```bash
bws project edit 1c80965c-acb3-486e-ac24-b03000dc7318 --name "My project 2"
```

By default, `bws project edit` will return objects as a JSON array, as in the following example. You can alter the output format using the `--output` flag (learn more).

```bash
{
  "object": "project",
  "id": "1c80965c-acb3-486e-ac24-b03000dc7318",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "name": "My project 2",
  "creationDate": "2023-06-29T13:22:37.942559Z",
  "revisionDate": "2023-06-29T13:31:07.927829Z"
}
```

project get

The `get` command retrieves a specific project which the logged-in machine account can access from your vault. Objects in your vault that the machine account does not have access to cannot be retrieved.

```bash
bws project get <PROJECT_ID>
```

To get a specific project, use the following command where e325ea69-a3ab-4dff-836f-b02e013fe530 represents a `PROJECT_ID`:

```bash
bws project get e325ea69-a3ab-4dff-836f-b02e013fe530
```

By default, `get` will return objects as a JSON array, as in the following example. You can alter the output format using the `--output` flag (learn more).

```bash
{
  "object": "project",
  "id": "e325ea69-a3ab-4dff-836f-b02e013fe530",
  "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
  "name": "App 1",
  "creationDate": "2023-06-27T19:24:42.181607Z",
  "revisionDate": "2023-06-27T19:24:42.181607Z"
}
```

project list

To list the projects this machine account has access to, use the following command:

```bash
bws project list
```

By default, `list` will return objects as a JSON array, as in the following example. You can alter the output format using the `--output` flag (learn more).

```bash
[
  {
    "object": "project",
    "id": "e325ea69-a3ab-4dff-836f-b02e013fe530",
    "organizationId": "10e8cbfa-7bd2-4361-bd6f-b02e013f9c41",
    "name": "App 1",
    "creationDate": "2023-06-27T19:24:42.181607Z",
    "revisionDate": "2023-06-27T19:24:42.181607Z"
  }.
  ...
]
```

config

The config command specifies server settings for the Secrets Manager CLI to use. A primary use of `bws config` is to connect the CLI to a self-hosted Bitwarden server.

server

Available `bws` server settings include `server-base`, `server-api`, and `server-identity`, for example:

```bash
bws config server-base https://my_hosted_server.com
```

Note

If `server_api` and `server_identity` are not configured, the values will default to the `server_base` value. For example:
`https://serverbase.com/api`
`https://serverbase.com/identity`

When done this way, your specified server values will be saved to a default profile in a `~/.config/bws/config` file. You can use subsequent options to create alternate profiles and config files:

config --profile

Use the `--profile` option with the `config` command to save specified server values to alternate profiles, for example:

```bash
bws config server-base http://other_hosted_server.com --profile dev
```

Once created, you can use that profile with other commands to route requests to the specified server, for example:

```bash
bws secret get 2863ced6-eba1-48b4-b5c0-afa30104877a --profile dev
```

config --config-file

Use the `--config-file` option with the `config` command to save specified server values to alternate config files, for example to save values to a default profile in a new config file:

```bash
bws config server-base http://third_hosted_server.com --config-file ~/.bws/alt_config
```

You can chain `--config-file` with `--profile` to save values to alternate profiles in alternate config files, for example:

```bash
bws config server-base http://third_hosted_server.com --config-file ~/.bws/alt_config --profile alt
_dev
```

Once created, you can use that profile with other commands to route requests to the specified server, for example:

```bash
bws secret get 2863ced6-eba1-48b4-b5c0-afa30104877a --config-file ~/.bws/alt_config --profile alt_d
ev
```

config --state

State files are fully encrypted files that store authentication tokens and additional relevant data. State files can reduce rate limiting while authenticating, using stored tokens for authentication. The state directory default location is ~/.config/bws/state. The state file must be designated with an absolute path:

```
bws config state-dir /Users/user/Desktop/bws/state
```

Users may opt out of using state files by accessing the ~/.config/bws/config and setting `state_opt_out` to the values `true` or `1`.

Config Docker

Pass config file into Docker container with run command:

Plain Text

```
docker run -it -v /PATH/TO/YOUR/CONFIGFILE:/home/app/.bws/config -e BWS_ACCESS_TOKEN=<ACCESS_TOKEN_
VALUE> bitwarden/bws secret list
```

Options

-o, --output

By default, the Secrets Manager CLI will return a JSON object or array of JSON objects in response to commands. Output format can be altered to fits your needs using the `-o`, `--output` flag along with one of the following options:

- `json`: Default. Output JSON.
- `yaml`: Output YAML.
- `table`: Output an ASCII table with keys as column headings.
- `tsv`: Output tab-separated values with no keys.
- `none`: Only output errors and warnings.
- `env`: Output secrets in KEY=VALUE format.

For example, the command:

Bash

```bash
bws secret get 2863ced6-eba1-48b4-b5c0-afa30104877a --output yaml
```

will return the following:

Bash

```bash
object: secret
id: 2863ced6-eba1-48b4-b5c0-afa30104877a
organizationId: b8824f88-c57c-4a36-8b1a-afa300fe0b52
projectId: 1d0a63e8-3974-4cbd-a7e4-afa30102257e
key: Stripe API Key
value: osiundfpowubefpouwef
note: 'These are notes.'
creationDate: 2023-02-08T15:48:33.470701Z
revisionDate: 2023-02-08T15:48:33.470702Z
```

Note

While using the env output format, if the key name is non-POSIX-compliant, that key value pair will be commented-out and a comment at the bottom of the output will be displayed indicating that the output has been modified.

Using the `--output env flag`, for example:

```bash
bws secret list --output env
```

will return the following:

```bash
this_is_a_keyname="this is a key value"
CLOUDFLARE_API_TOKEN="1234123412341234123412341234"
# This is an invalid keyname="this will get commented-out"

# one or more secrets have been commented-out due to a problematic key name
```

-c, --color

Output can further be customized by indicated whether you would like colorized output. Available values for this option are `yes`, `no`, and `auto`.

--access-token

You can authenticate individual CLI requests using the `-t`, `--access-token` option with any individual command, for example:

```bash
bws secret list --access-token 0.48c78342-1635-48a6-accd-afbe01336365.C0tMmQqHnAp1h0gL8bngprlPOYutt
0:B3h5D+YgLvFiQhWkIq6Bow==
```

--profile

Use the `--profile` option with the `list` or `get` commands to specify which profile to use, for example:

```bash
bws secret get 2863ced6-eba1-48b4-b5c0-afa30104877a --profile dev
```

Refer to the `config` command (here) for help understanding and setting up alternate profiles.

--config-file

Use the `--config-file` option with the `--profile` option and `list` or `get` commands to specify which profile from which configuration file to use, for example:

```bash
bws secret get 2863ced6-eba1-48b4-b5c0-afa30104877a --config-file ~/.bws/alt_config --profile alt_d
ev
```

Refer to the `config` command (here) for help understanding and setting up alternate config files and profiles.

--server-url

This option can be used to set the server URL that the CLI will send the request associated with a given command to, for example:

```bash
bws list secrets --server-url http://my_hosted_server.com
```

This option will override any URLS configured via the `config` command (see here).

--help

Use this option to print help for any given `bws` command.

--version

Use this option to print the version of the `bws` client you're using.