dump-env

Release 1.5.0

Nikita Sobolev

Mar 22, 2024

CONTENTS

`dump-env` takes an `.env.template` file and some optional environmental variables to create a new `.env` file from these two sources. No external dependencies are used.

CHAPTER

ONE

WHY?

Why do we need such a tool? Well, this tool is very helpful when your CI is building `docker` (or other) images. Previously we had some complex logic of encrypting and decrypting files, importing secret keys and so on. Now we can just create secret variables for our CI, add some prefix to it, and use `dump-env` to make our life easier.

CHAPTER

TWO

INSTALLATION

```
$ pip install dump-env
```

CHAPTER

THREE

QUICKSTART

This quick demo will demonstrate the main and the only purpose of `dump-env`:

```
$ dump-env --template=.env.template --prefix='SECRET_ENV_' > .env
```

This command will:

1. take `.env.template`
2. parse its keys and values
3. read all the variables from the environment starting with `SECRET_ENV_`
4. remove this prefix
5. mix it all together, environment vars may override ones from the template
6. sort keys in alphabetic order
7. dump all the keys and values into the `.env` file

CHAPTER

FOUR

ADVANCED USAGE

4.1 Multiple prefixes

```
$ dump-env -t .env.template -p 'SECRET_ENV_' -p 'ANOTHER_SECRET_ENV_' > .env
```

This command will do pretty much the same thing as with one prefix. But, it will replace multiple prefixes. Further prefixes always replace previous ones if they are the same. For example:

```
$ export SECRET_TOKEN='very secret string'
$ export SECRET_ANSWER='13'
$ export ANOTHER_SECRET_ENV_ANSWER='42'
$ export ANOTHER_SECRET_ENV_VALUE='0'
$ dump-env -p SECRET_ -p ANOTHER_SECRET_ENV_
ANSWER=42
TOKEN=very secret string
VALUE=0
```

4.2 Strict env variables

In case you want to be sure that `YOUR_VAR` exists in your environment when dumping, you can use `--strict` flag:

```
$ dump-env --strict YOUR_VAR -p YOUR_
Missing env vars: YOUR_VAR
```

Oups! We forgot to create it! Now this will work:

```
$ export YOUR_VAR='abc'
$ dump-env --strict YOUR_VAR -p YOUR_
VAR=abc
```

Any number of `--strict` flags can be provided. No more forgotten template overrides or missing env vars!

4.3 Source templates

You can use an env template as a source template by using the `-s` or `--source` argument. This will restrict any non-prefixed variables found in the environment to only those already defined in your template.

```
$ cat template.env
ANSWER=13
TOKEN=very secret string
VALUE=0
```

```
$ export ANSWER='42'
$ dump-env --source=template.env
ANSWER=42
TOKEN=very secret string
VALUE=0
```

You can still also use prefixes to add extra variables from the environment

```
$ export EXTRA_VAR='foo'
$ dump-env -s template.env -p EXTRA_
ANSWER=13
TOKEN=very secret string
VALUE=0
VAR=foo
```

4.3.1 Strict Source

Using the `--strict-source` flag has the same effect as defining a `--strict` flag for every variable defined in the source template.

```
$ export ANSWER='42'
$ dump-env -s template.env --strict-source
Missing env vars: TOKEN, VALUE
```

CHAPTER

FIVE

CREATING SECRET VARIABLES IN SOME CIS

- travis docs
- gitlab-ci docs
- github actions

CHAPTER

SIX

REAL-WORLD USAGES

Projects that use this tool in production:

- wemake-django-template
- wemake-vue-template

CHAPTER

SEVEN

RELATED

You might also be interested in:

- https://github.com/wemake-services/dotenv-linter

CHAPTER

EIGHT

LICENSE

MIT

CHAPTER

NINE

API REFERENCE

dump(*template: str = ''*, *prefixes: List[str] | None = None*, *strict_keys: Set[str] | None = None*, *source: str = ''*, *strict_source: bool = False*) → Dict[str, str]

This function is used to dump `.env` files.

As a source you can use both: 1. env.template file (`''` by default) 2. env vars prefixed with some prefix (`''` by default)

Parameters

- **template** – The path of the `.env` template file, use an empty string when there is no template file.
- **prefixes** – List of string prefixes to use only certain env variables, could be an empty string to use all available variables.
- **strict_keys** – List of keys that must be presented in env vars.
- **source** – The path of the `.env` template file, defines the base list of env vars that should be checked, disables the fetching of non-prefixed env vars, use an empty string when there is no source file.
- **strict_source** – Whether all keys in source template must also be presented in env vars.

Returns

Ordered key-value pairs of dumped env and template variables.

Raises

StrictEnvError – when some variable from template is missing.

main() → NoReturn

Runs dump-env script.

Example:

```
This example will dump all environ variables:

.. code:: bash

    $ dump-env

This example will dump all environ variables starting with ``PIP_``:

.. code:: bash

    $ dump-env -p 'PIP_'
```

(continues on next page)

(continued from previous page)

```
This example will dump all environ variables starting with ``PIP_``
and update them with variables starting with ``SECRET_``:

.. code:: bash

    $ dump-env -p 'PIP_' -p 'SECRET_'

This example will dump everything from ``.env.template`` file
and all env variables with ``SECRET_`` prefix into a ``.env`` file:

.. code:: bash

    $ dump-env -p 'SECRET_' -t .env.template > .env

This example will fail if ``REQUIRED`` does not exist in environ:

.. code:: bash

    $ dump-env --strict=REQUIRED

This example will dump everything from a source ``.env.template`` file
with only env variables that are defined in the file:

.. code:: bash

    $ dump-env -s .env.template

This example will fail if any keys in the source template do not exist
in the environment:

.. code:: bash

    $ dump-env -s .env.template --strict-source
```

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

D

M