neze-webcli Documentation

neze

Aug 13, 2019

Contents

CHAPTER 1

Installation

1.1 With PyPi

The `neze-webcli` project is available on pypi.

```
pip install --upgrade neze-webcli
```

Note that you can also install the development version.

```
pip install --upgrade --pre neze-webcli
```

1.2 From source

Just install the requirements listed in `requirements.txt`, either by hand or with `pip`.

```
pip install -r requirements.txt
```

1.3 Development setup

Best way to setup your development is probably with a virtual environment. I use a virtualenv set up with virtualenv-wrapper.

```
cd /path/to/webcli/source
mkvirtualenv -p $(which python3) -a $(pwd) webcli
pip install -e .
```

Note that if you want to build documentation you will need to install sphinx.

```
workon webcli
pip install -r requirements.dev.txt
python setup.py doc
```

CHAPTER 2

Configuration files

See *Configuration structures*

CHAPTER 3

The WEBCLI python3 package

3.1 API

3.1.1 API Interface

The `webcli.api` module provides an interface for implementing API functions and APIs.

3.1.1.1 Usage example

Let's start by defining one API function class.

```
class XFunction(APIFunction):
    # Default method will be get.
    __call__=APIFunction.get

    def _prepare(self,rq):
        rq.url += self.path
        if 'id' in rq.kwargs:
            rq.url += ('/%d' % rq.kwargs['id'])
            del rq.kwargs['id']

    def _prepare_post(self,rq):
        self._prepare(rq)
        rq.request.update(params=rq.kwargs)

    def _process(self,r):
        r.retry = False
```

Then we define the API with its functions.

```
class XAPI(API):
    def __init__(self,url,token):
```

(continues on next page)

(continued from previous page)

```
        super().__init__(url)
        self._token=token
        self['/counters'] = XFunction()

    def _prepare(self,rq):
        rq.request.update(headers={'private-token': self._token})
        rq.url = self.url

    def _process(self,r):
        r.raise_for_status()
        r.data = r.json()
```

Instanciating the API and calling methods is then pretty straightforward.

```
api = XAPI('https://example.com/api','c11d5f2ddd2fc3fe')

# Get list of counters
result = api['/counters']()
result = api['/counters'].get()
# Get counter number 42
result = api['/counters'](id=42)
# Reset counter number 42 to 0
result = api['/counters'].post(id=42,value=0)
```

When calling an API function, the request goes through processing:

- Request is processed by the `API._prepare()` method. `API._prepare()` is the default method if `API._prepare_@method()` is not defined.
- Then, it is processed by the same method of the API function: `APIFunction._prepare()` by default if not `APIFunction._prepare_@method()`
- Request is sent by the `requests` module.
- The response is processed by the `API._process()` method. `API._process()` is the default method if `API._process_@method()` is not defined.
- Then, the response is processed by the same method of the API function: `APIFunction._process()` by default if not `APIFunction._process_@method()`

This process is handled by the `webcli.api.APIFunction.request()` method.

3.1.1.2 API Functions

One API function is used to send a HTTP request.

3.1.1.3 API

3.1.2 Gitlab API

3.1.3 Transmission API

3.2 CLI

3.2.1 CLI Interface

3.2.2 CLI Patches

3.2.2.1 Display patch

3.2.2.2 Config patch

3.2.3 Git Piptag CLI

3.2.3.1 Usage

See *Git PIPTAG*

```
usage: git-piptag [-h] [-r | -s | -d | -g] [-n | -f] [tag]

Manages git tags for pypi versioning. See PEP440.

optional arguments:
  -h, --help  show this help message and exit

action:
  -r, --root  Get and print the latest version tag among the parents commits
              in the git tree. This is the default operation.
  -s, --set   Set version tag in git. This is usually done for a release. Use
              the dry-run mode (-n) before actually running this (-f).
  -d, --dev   Set .dev0 version tag in git. `git piptag -fd 3.0` means that
              your next commits will be tagged v3.0.devN meaning that version
              3.0 is the next release being developed. Do not specify a `.dev`
              version yourself in the command line. Use the dry-run mode (-n)
              before actually running this (-f).
  -g, --get   Get and print the current version tag by offsetting the root
              tag.

dry run:
  If none of these is selected, the `root` and `get` functions are usable.
  Selecting one of these two modes switches to the `set` and `dev`
  functions.

  -n          Dry Run
  -f          Actually Run

tag:
  Every tags used by this program follow the PEP440 specification. It is
  available at https://www.python.org/dev/peps/pep-0440/
```

(continues on next page)

(continued from previous page)

```
tag         Your proposal for a version tag. Should not be a development or
            local tag.
```

3.2.3.2 Module contents

3.2.4 Gitlab CLI

3.2.5 Transmission CLI

3.3 Data management

3.3.1 Configuration structures

The `webcli.data.config` provides interfaces for working with configuration files.

3.3.1.1 Configuration files

Usually, there are several configuration levels, in a git-like way. For example the default `webcli.data.config.FileNames` object defines three levels:

- `system` corresponding to `/etc/<name>.<ext>` configuration files
- `global` corresponding to `~/.<name>.<ext>` or `~/.config/<name>.<ext>`
- `local` corresponding to `$(git-dir)/<name>.<ext>`

It supports the default extensions, currently `json`, `yaml` and `ini`.

The levels are defined in order just like in git, which means that in this example `global` values override `system` values.

3.3.1.2 Configuration content

Every configuration value is of the form `<section>.<key>=<value>`, with a special `DEFAULT` section, also shortened as `<key>=<value>`.

Examples

```
[DEFAULT]
foo=42

[prod]
foo=13
```

```
{
  "DEFAULT": {"foo":42},
  "prod": {"foo":13}
}
```

```yaml
DEFAULT:
  foo: 42
prod:
  foo: 13
```

Special keys

Special configuration keys use other modules to fetch more information.

- `@secrets`: see *Secrets management*

```
[DEFAULT]
@secrets=pass://www/example.com
```

3.3.1.3 Configuration specification

The `webcli.data.config.Spec` class is used to define configuration content.

A configuration finding unknown keys or sections will then refuse the faulty file.

By defining the `DEFAULT` section, however, it autorizes sections with arbitrary names that will have to follow the specification of the default section.

3.3.1.4 Configuration example

```python
spec = Spec('service')
spec.add_section('DEFAULT')
spec.add_key('@secrets',str)
spec.add_key('user',str)
spec.add_key('password',str)

config = Config(spec)

# Write configuration value in the cache AND in the local configuration file
config['local.test.user'] = 'anonymous'

config['global.test.user'] = 'me'

print(config['test.user'])
# 'anonymous'
```

3.3.1.5 Python module

3.3.2 Cookie objects

3.3.3 File storage

3.3.4 Secrets management

3.4 Utility tools

Todo: Move code in correct places

3.4.1 Choices lists

3.4.2 Docker-related configurations

3.4.3 Custom iterators

3.4.4 Read-only dictionaries

CHAPTER 4

Binaries

4.1 Git PIPTAG

4.1.1 Usage

The `git-piptag` executable simply calls the module `webcli.cli.git_piptag`.

```
which git-piptag || alias git-piptag="python3 -m webcli.cli.git_piptag"
```

This program has four modes:

root Finds the latest version tag in the git tree.

```
git piptag [--root|-r]
```

get Get the tag of the HEAD. If you do not propose a tag, it is automatically calculated depending on the latest version tag and the distance with it in the git tree. Using get with a proposed-tag will only have as an effect to parse and normalize your tag.

```
git piptag [--get|-g] [proposed-tag]
```

set Put the tag of the HEAD in the tree. In the default dry-run mode (`-n`) it just prints the git command that would be executed. In force mode (`-f`) the tag is set by calling `git tag`.

```
git piptag [--set|-s] [-n|-f] [proposed-tag]
```

dev Put a `dev0` tag in the tree in the same way as **set**. The intended meaning is that from this point in the tree you are doing development commits for your next version. If you do not propose a tag (of a stable next version), the next version number is calculated automatically from the current one. This computation is not guaranteed to produce coherent output in some git tree states.

```
git piptag [--dev|-d] [-n|-f] [proposed-tag]
```

4.1.2 Workflow Example

```
λ git init .
λ touch .gitignore
λ git add .gitignore && git commit -m "init"

  * 6e844c9 (HEAD -> master) init

λ git piptag # Default root tag is the lowest possible and not very interesting.
0a0.dev0

λ touch v1 && git add v1 && git commit -m "release 1"

  * 669e09e (HEAD -> master) release 1
  * 6e844c9 init

λ git piptag v1.0 -n # We would like to release v1.0. Start with a dry run.
git tag -s -m 'Automatic v1.0 Tag by Git Piptag' v1.0

λ git piptag v1.0 -f # We're satisfied. Let's apply this.

  * 669e09e (HEAD -> master, tag: v1.0) release 1
  * 6e844c9 init

λ git piptag -g # What would be the tag of the current commit now?
1.0+git.669e09eb

λ git checkout -b dev
λ touch todo && git add todo && git commit -m "v2 plan"

  * 5b1c624 (HEAD -> dev) v2 plan
  * 669e09e (tag: v1.0, master) release 1
  * 6e844c9 init

λ git piptag -g # Wrongly gives a 'post v1' tag while we're developing v2
1.0.post1+git.5b1c6243

λ git piptag -d 2.0 -n # Say that we're developing v2
git tag -s -m 'Automatic v2.0.dev0 Tag by Git Piptag' v2.0.dev0

λ git piptag -d 2.0 -f

  * 5b1c624 (HEAD -> dev, tag: v2.0.dev0) v2 plan
  * 669e09e (tag: v1.0, master) release 1
  * 6e844c9 init

λ touch feature && git add feature && git commit -m "new feature"

  * 70f8129 (HEAD -> dev) new feature
  * 5b1c624 (tag: v2.0.dev0) v2 plan
  * 669e09e (tag: v1.0, master) release 1
  * 6e844c9 init

λ git piptag -g # The tag of the current commit is now more coherent
2.0.dev1+git.70f8129c

λ touch v2b && git add v2b && git commit -m "beta release 2"
```

(continues on next page)

(continued from previous page)

```
λ git piptag 2.0b -f # Tell we're releasing a beta

  * 985edf0 (HEAD -> dev, tag: v2.0b0) beta release 2
  * 70f8129 new feature
  * 5b1c624 (tag: v2.0.dev0) v2 plan
  * 669e09e (tag: v1.0, master) release 1
  * 6e844c9 init

λ git checkout master
λ touch fix && git add fix && git commit -m "bugfix"

  * 10710b4 (HEAD -> master) bugfix
  | * 985edf0 (tag: v2.0b0, dev) beta release 2
  | * 70f8129 new feature
  | * 5b1c624 (tag: v2.0.dev0) v2 plan
  |/
  * 669e09e (tag: v1.0) release 1
  * 6e844c9 init

λ git piptag -g # The tag is now post-release
1.0.post1+git.10710b40
```

4.2 Transmission

4.2.1 Usage

The `transmission` executable simply calls the module `webcli.cli.transmission` to interact with a remote transmission API.

```
which transmission || alias transmission="python3 -m webcli.cli.transmission"
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search