temple Documentation

Release 1.4.2

Clover Health

Jul 31, 2018

Contents

CHAPTER 1

Temple - Templated Project Management

Temple provides templated project creation and management.

The main functionality of temple includes:

1. Creating new projects from cookiecutter templates.
2. Listing all available templates under a github user / org along with all projects created from those templates.
3. Keeping projects up to date with the template as it changes.

A quick start is provided below. Be sure to go through the *Installation* section before starting. It's also useful to read about cookiecutter templates since they form the foundation of this tool.

1.1 Quick Start

1.1.1 Listing templates and projects

Temple manages projects that are started from templates (specifically, cookiecutter templates). In order to see what templates are available for use within your Github org, do:

```
temple ls <github_org_name>
```

This will list all of the paths of templates that are available. Doing:

```
temple ls <github_org_name> -l
```

will also display the description of the template.

To list all projects created with a template (and the project's descriptions), take the template path from `temple ls` and use it as an argument like so:

```
temple ls <github_org_name> <git@github.com:user/cookiecutter-template.git> -l
```

1.1.2 Starting new projects

A new project can be set up from a template with:

```
temple setup <git@github.com:user/cookiecutter-template.git>
```

What happens next is dependent on how the template is configured. By default, cookiecutter will prompt the user for template parameters, defined in the `cookiecutter.json` file of the template repository. If any cookiecutter hooks are defined in the project, additional setup steps will happen that are specific to the type of project being started.

1.1.3 Keeping your project up to date with the latest template

Once a project is set up and published to Github, temple-created projects can be listed with the `temple ls` command. If a template is ever updated, changes can be pulled into a project with:

```
temple update
```

This will git merge the template changes into your repository. You will need to review the changes, resolve conflicts, and then `git add` and `git push` these changes yourself.

Sometimes it is desired that projects always remain up to date with the latest template - for example, ensuring that each project obtains a security patch to a dependency or doing an organization-wide upgrade to a new version of Python.

Using `temple update --check` from the repository will succeed if the project is up to date with the latest template or return a non-zero exit code if it isn't. This command can be executed as part of automated testing that happens in continuous integration in order to ensure all projects remain up to date with changes before being deployed.

Note: Updating your project with the latest template does not result in cookiecutter hooks being executed again.

1.1.4 Switching your project to another template

Sometimes it is desirable to switch a project to another template, like when open sourcing a private package. Projects can be switched to another template with:

```
temple switch <git@github.com/user/new-template-path.git>
```

Similar to `temple update`, you will need to review the changes, resolve conflicts, and then `git add` and `git push` these changes.

Note: Switching templates does not trigger any cookiecutter hooks. Users must manually do any project setup and must similarly do any project teardown that might have resulted from the previously template. The authors have intentionally left out this convenience for now since temple currently has no way to spin down projects.

1.2 Next Steps

For more detailed docs about the temple command line interface (CLI) and python package, view the *Temple CLI* and *Temple Package* sections.

Want to create your own temple-managed project? View the *Creating Templates Managed by Temple* section.

CHAPTER 2

Installation

temple can be installed with:

```
pip3 install temple
```

Most temple functionality requires a `GITHUB_API_TOKEN` environment variable to be set. The Github API token is a personal token that you create by following the Github Access Token Instructions. This token only requires `repo` scope.

Note: Temple requires a Github API token for listing available templates, starting new projects, and updating a project with a template. However, project templates themselves might have other setup requirements. Consult the documentation of templates you want to use for your projects for information about other installation and setup required.

CHAPTER 3

Creating Templates Managed by Temple

Under the hood, temple uses cookiecutter to gather user input about a project and then spin up a local directory with the project scaffolding. In order to learn more about how to make your own cookiecutter template, consult the cookiecutter docs.

After you have created a template and published it to Github, it will be displayed with `temple ls` and can also be used by `temple setup`. There is no additional setup required. Make sure the description of your repo is filled in on Github, because that will be returned when users type `temple ls -l`.

3.1 Making Project Creation Seamless with Cookiecutter Hooks

Once you have created a cookiecutter template and published it to Github, it will work with temple out of the box, but there are ways to make project setup even more seamless.

For example, say a cookiecutter template has been created at git@github.com:user/cookiecutter-template.git. When the user calls `temple setup git@github.com:user/cookiecutter-template.git`, the templated project will be created locally, but the user will be left to do remaining setup steps manually (like pushing to Github, setting up continuous integration, etc).

Cookiecutter offers the ability to insert pre or post generate hooks before and after a project is created, allowing project-specific setup steps to happen. Some of the examples given in the hook docs include ensuring a python module name is valid.

Hooks can be used for initial project setup in a variety of ways, some examples including:

1. Creating a remote github repository for the project
2. Pushing to a remote github repository after the project is created
3. Adding default collaborators to a project
4. Setting up continuous integration for a project
5. Creating an initial server for a web app along with a domain name

Keep in mind that cookiecutter hooks are called during `temple setup` and `temple update`. Although hooks should be idempotent in the case of transient setup failures, sometimes it is not desirable to have hooks execute during `setup` and `update`. In order to customize this behavior in your hooks, temple exports the `_TEMPLE` environment variable and sets it to value of the command being executed (i.e. "ls", "setup", or "update").

Below is an example of creating a `pre_gen_project.py` hook in the `hooks` directory of the template. The script ensures that the cookiecutter template is only used by temple and not by `cookiecutter` or another templating library:

```
#!/usr/bin/env python

import os

if __name__ == "__main__":
    if not os.environ.get('_TEMPLE'):
        raise Exception('This template can only be used by temple')
```

Here's an example of pushing the newly-created project to github with a `post_gen_project.py` file:

```
#!/usr/bin/env python3

import os
import subprocess

def call_cmd(cmd, check=True):
    """Call a command. If check=True, throw an error if command fails"""
    return subprocess.call(cmd, check=check)

def push_to_github():
    call_cmd('git init')
    call_cmd('git add .')
    call_cmd('git commit -m "Initial project scaffolding"')
    # Use the "repo_name" template variable
    call_cmd('git remote add origin {repo_name}')
    ret = call_cmd('git push origin master', check=False)
    if ret.returncode != 0:
        # Do additional error handling if the repo already exists.
        # Maybe the user already created the remote repository..
        pass

if __name__ == "__main__":
    # Only run these commands when "temple setup" is being called
    if os.environ.get('_TEMPLE') == 'setup':
        push_to_github()
```

In the above hook, the `push_to_github` function is called only when running `temple setup`. In other words, this hook code will not run on `temple update` or any other commands that invoke the template to be rendered.

As shown above, variables that are part of the template, like `{repo_name}` can be referenced and used in the hooks. If the above fails, it will cause all of `temple setup` to fail, which will in turn not create any local project on the user's machine. Idempotency of project hooks should be kept in mind when designing them.

Note: The hooks shown can also be written in shell and named `pre_gen_project.sh` and `post_gen_project.sh`.

CHAPTER 4

Temple CLI

The main `temple` command and its subcommands are listed below. Note that `--help` can be given as an argument to any of these commands to print out help on the command line.

4.1 temple

```
temple [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show version

4.1.1 clean

Cleans temporary resources created by temple, such as the temple update branch

```
temple clean [OPTIONS]
```

4.1.2 ls

List packages created with temple. Enter a github user or organization to list all templates under the user or org. Using a template path as the second argument will list all projects that have been started with that template.

Use "-l" to print the Github repository descriptions of templates or projects.

```
temple ls [OPTIONS] GITHUB_USER [TEMPLATE]
```

Options

-l, --long-format
Print extended information about results

Arguments

GITHUB_USER
Required argument

TEMPLATE
Optional argument

4.1.3 setup

Setup new project. Takes a full git SSH path to the template as returned by "temple ls". In order to start a project from a particular version (instead of the latest), use the "-v" option.

```
temple setup [OPTIONS] TEMPLATE
```

Options

-v, --version <version>
Git SHA or branch of template to use for creation

Arguments

TEMPLATE
Required argument

4.1.4 switch

Switch a project's template to a different template.

```
temple switch [OPTIONS] TEMPLATE
```

Options

-v, --version <version>
Git SHA or branch of template to use for update

Arguments

TEMPLATE
Required argument

4.1.5 update

Update package with latest template. Must be inside of the project folder to run.

Using "-e" will prompt for re-entering the template parameters again even if the project is up to date.

Use "-v" to update to a particular version of a template.

Using "-c" will perform a check that the project is up to date with the latest version of the template (or the version specified by "-v"). No updating will happen when using this option.

```
temple update [OPTIONS]
```

Options

-c, --check
Check to see if up to date

-e, --enter-parameters
Enter template parameters on update

-v, --version <version>
Git SHA or branch of template to use for update

CHAPTER 5

Temple Package

5.1 temple.setup

Creates and initializes a project from a template

temple.setup.**setup**(*template*, *version=None*)
Sets up a new project from a template

Note that the *temple.constants.TEMPLE_ENV_VAR* is set to 'setup' during the duration of this function.

Parameters

- **template** (*str*) – The git SSH path to a template
- **version** (*str, optional*) – The version of the template to use when updating. Defaults to the latest version

5.2 temple.update

Updates a temple project with the latest template

temple.update.**up_to_date**(*version=None*)
Checks if a temple project is up to date with the repo

Note that the *temple.constants.TEMPLE_ENV_VAR* is set to 'update' for the duration of this function.

Parameters **version** (*str, optional*) – Update against this git SHA or branch of the template

Returns True if up to date with `version` (or latest version), False otherwise

Return type boolean

Raises

- *NotInGitRepoError* – When running outside of a git repo

- *InvalidTempleProjectError* – When not inside a valid temple repository

temple.update.**update**(*old_template=None*, *old_version=None*, *new_template=None*, *new_version=None*, *enter_parameters=False*)

Updates the temple project to the latest template

Proceeeds in the following steps:

1. Ensure we are inside the project repository
2. Obtain the latest version of the package template
3. If the package is up to date with the latest template, return
4. If not, create an empty template branch with a new copy of the old template
5. Create an update branch from HEAD and merge in the new template copy
6. Create a new copy of the new template and merge into the empty template branch
7. Merge the updated empty template branch into the update branch
8. Ensure temple.yaml reflects what is in the template branch
9. Remove the empty template branch

Note that the *temple.constants.TEMPLE_ENV_VAR* is set to 'update' for the duration of this function.

Two branches will be created during the update process, one named `_temple_update` and one named `_temple_update_temp`. At the end of the process, `_temple_update_temp` will be removed automatically. The work will be left in `_temple_update` in an uncommitted state for review. The update will fail early if either of these branches exist before the process starts.

Parameters

- **old_template** (*str, default=None*) – The old template from which to update. Defaults to the template in temple.yaml
- **old_version** (*str, default=None*) – The old version of the template. Defaults to the version in temple.yaml
- **new_template** (*str, default=None*) – The new template for updating. Defaults to the template in temple.yaml
- **new_version** (*str, default=None*) – The new version of the new template to update. Defaults to the latest version of the new template
- **enter_parameters** (*bool, default=False*) – Force entering template parameters for the project

Raises

- *NotInGitRepoError* – When not inside of a git repository
- *InvalidTempleProjectError* – When not inside a valid temple repository
- *InDirtyRepoError* – When an update is triggered while the repo is in a dirty state
- *ExistingBranchError* – When an update is triggered and there is an existing update branch

Returns True if update was performed or False if template was already up to date

Return type boolean

5.3 temple.ls

Lists all temple templates and projects spun up with those templates

temple.ls.**ls**(*github_user*, *template=None*)
Lists all temple templates and packages associated with those templates

If `template` is None, returns the available templates for the configured Github org.

If `template` is a Github path to a template, returns all projects spun up with that template.

`ls` uses the github search API to find results.

Note that the *temple.constants.TEMPLE_ENV_VAR* is set to 'ls' for the duration of this function.

Parameters

- **github_user** (*str*) – The github user or org being searched.
- **template** (*str, optional*) – The template git repo path. If provided, lists all projects that have been created with the provided template. Note that the template path is the SSH path (e.g. git@github.com:CloverHealth/temple.git)

Returns A dictionary of repository information keyed on the SSH Github url

Return type dict

Raises *InvalidGithubUserError* – When `github_user` is invalid

5.4 temple.constants

Constants for temple

temple.constants.**GITHUB_API_TOKEN_ENV_VAR = 'GITHUB_API_TOKEN'**
The Github API token environment variable

temple.constants.**TEMPLE_CONFIG_FILE = 'temple.yaml'**
The temple config file in each repo

temple.constants.**TEMPLE_DOCS_URL = 'https://github.com/CloverHealth/temple'**
Temple docs URL

temple.constants.**TEMPLE_ENV_VAR = '_TEMPLE'**
The environment variable set when running any temple command. It is set to the name of the command

temple.constants.**UPDATE_BRANCH_NAME = '_temple_update'**
The temporary branches used for updates

5.5 temple.exceptions

Temple exceptions

exception temple.exceptions.**CheckRunError**
When running `temple update --check` errors

exception temple.exceptions.**Error**
The top-level error for temple

exception temple.exceptions.**ExistingBranchError**
Thrown when a specifically named branch exists or doesn't exist as expected.

exception temple.exceptions.**InDirtyRepoError**
Thrown when running in a dirty git repo

exception temple.exceptions.**InGitRepoError**
Thrown when running inside of a git repository

exception temple.exceptions.**InvalidCurrentBranchError**
Thrown when a command cannot run because of the current git branch

exception temple.exceptions.**InvalidEnvironmentError**
Thrown when required environment variables are not set

exception temple.exceptions.**InvalidGithubUserError**
An invalid github user was passed to ls.

exception temple.exceptions.**InvalidTemplatePathError**
Thrown when a template path is not a Github SSH path

exception temple.exceptions.**InvalidTempleProjectError**
Thrown when the repository was not created with temple

exception temple.exceptions.**NotInGitRepoError**
Thrown when not running inside of a git repo

exception temple.exceptions.**NotUpToDateWithTemplateError**
Thrown when a temple project is not up to date with the template

CHAPTER 6

Future Work

1. Use python wrappers for git and Github access (GitPython, github3.py)

CHAPTER 7

Release Notes

```
1.4.3
-----

* [DI-8] Temple update 2018-07-31 (#5)

1.4.2
-----

* Added primary authors section (#4)

1.4.1
-----

* Added ReadTheDocs requirements file

1.4.0
-----

* sem-ver: api-break, Initial open source release of temple (#1)
* first commit
```

CHAPTER 8

Contributing Guide

This project was created using temple. For more information about temple, go to the Temple docs.

8.1 Setup

Set up your development environment with:

```
git clone git@github.com:CloverHealth/temple.git
cd temple
make setup
```

`make setup` will setup a virtual environment managed by pyenv and install dependencies.

Note that if you'd like to use something else to manage dependencies other than pyenv, call `make dependencies` instead of `make setup`.

8.2 Testing and Validation

Run the tests with:

```
make test
```

Validate the code with:

```
make validate
```

8.3 Documentation

Sphinx documentation can be built with:

```
make docs
```

The static HTML files are stored in the `docs/_build/html` directory. A shortcut for opening them is:

```
make open_docs
```

8.4 Releases and Versioning

Anything that is merged into the master branch will be automatically deployed to PyPI. Documentation will be published to ReadTheDocs.

The following files will be generated and should *not* be edited by a user:

- `ChangeLog` - Contains the commit messages of the releases. Please have readable commit messages in the master branch and squash and merge commits when necessary.
- `AUTHORS` - Contains the contributing authors.
- `version.py` - Automatically updated to include the version string.

This project uses Semantic Versioning through PBR. This means when you make a commit, you can add a message like:

```
sem-ver: feature, Added this functionality that does blah.
```

Depending on the sem-ver tag, the version will be bumped in the right way when releasing the package. For more information, about PBR, go the the PBR docs.

Python Module Index

t

Index

Symbols

C

E

G

I

L

N

S

T

U