prereceivecli Documentation

Release 1.1.3

Costas Tyfoxylos

Jun 08, 2021

Contents

Contents:

CHAPTER 1

prereceivecli

A cli that implements a gitlab git server side pre-receive hook that gets driven from dynamodb and reports to slack offending pushes.

- Documentation: https://prereceivecli.readthedocs.org/en/latest

1.1 Development Workflow

The workflow supports the following steps

- lint
- test
- build
- document
- upload
- graph

These actions are supported out of the box by the corresponding scripts under _CI/scripts directory with sane defaults based on best practices. Sourcing setup_aliases.ps1 for windows powershell or setup_aliases.sh in bash on Mac or Linux will provide with handy aliases for the shell of all those commands prepended with an underscore.

The bootstrap script creates a .venv directory inside the project directory hosting the virtual environment. It uses pipenv for that. It is called by all other scripts before they do anything. So one could simple start by calling _lint and that would set up everything before it tried to actually lint the project

Once the code is ready to be delivered the _tag script should be called accepting one of three arguments, patch, minor, major following the semantic versioning scheme. So for the initial delivery one would call

```
$ _tag –minor
```

which would bump the version of the project to 0.1.0 tag it in git and do a push and also ask for the change and automagically update HISTORY.rst with the version and the change provided.

So the full workflow after git is initialized is:

- repeat as necessary (of course it could be test - code - lint :))
  - code
  - lint
  - test
- commit and push
- develop more through the code-lint-test cycle
- tag (with the appropriate argument)
- build
- upload (if you want to host your package in pypi)
- document (of course this could be run at any point)

1.2 Important Information

This template is based on pipenv. In order to be compatible with requirements.txt so the actual created package can be used by any part of the existing python ecosystem some hacks were needed. So when building a package out of this **do not** simple call

```
$ python setup.py sdist bdist_egg
```

as this will produce an unusable artifact with files missing. Instead use the provided build and upload scripts that create all the necessary files in the artifact.

1.3 Project Features

Can protect directories and files from tampering by checking hash entries for them in a dynamodb Please refer to USAGE.rst for setup details.

CHAPTER 2

Installation

At the command line:

```
$ pip install prereceivecli
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv prereceivecli
$ pip install prereceivecli
```

Or, if you are using pipenv:

```
$ pipenv install prereceivecli
```

CHAPTER 3

Usage

To develop on prereceivecli:

```
# The following commands require pipenv as a dependency

# To lint the project
_CI/scripts/lint.py

# To execute the testing
_CI/scripts/test.py

# To create a graph of the package and dependency tree
_CI/scripts/graph.py

# To build a package of the project under the directory "dist/"
_CI/scripts/build.py

# To see the package version
_CI/scipts/tag.py

# To bump semantic versioning [--major|--minor|--patch]
_CI/scipts/tag.py --major|--minor|--patch

# To upload the project to a pypi repo if user and password are properly provided
_CI/scripts/upload.py

# To build the documentation of the project
_CI/scripts/document.py
```

To use prereceivecli in a project:

The convention is that dynamodb holds a table with the name {parent_project}_git_hook and entries like

```
{u'protected_items': [{u'hashes': [sha1_hash],
                       u'name': <NAME>,
```

(continues on next page)

(continued from previous page)

```
                        u'type': <file | directory>}],
u'slug': <PROJECT_NAME>})
```

So a group in gitlab called "code" with a project called "super-secret" would need an entry in dynamodb in a table called "code_git_hook" and an entry like

```
{u'protected_items': [{u'hashes': ['asdfsfahjsfdhfhh134h234h23ghhhhqe3rh'],
                       u'name': '_CI',
                       u'type': 'directory'}],
u'slug': 'super-secret'}
```

if one wanted to protect the _CI directory of the project from tampering.

At least python3.6 is required.

```
pip install prereceivecli

# make system directory for configuration
sudo mkdir /etc/prereceive

# copy over the logging.json from the project
sudo cp /usr/local/lib/{PYTHON_VERSION_HERE}/site-packages/prereceivecli/conf/logging.
↪json /etc/prereceive/logging.json

# create the calling script as "root"
cat <<EOF > /etc/prereceive/pre-receive_active
#!/bin/sh
SLACK_WEB_HOOK=SLACK_WEBHOOK
AWS_SECRET=AWS_SECRET
AWS_KEY=AWS_KEY
AWS_REGION=eu-west-1

/usr/local/bin/pre-receive -l /etc/prereceive/logging.json \
                           -w "${SLACK_WEB_HOOK}" \
                           -s "${AWS_SECRET}" \
                           -k "${AWS_KEY}"  \
                           -r "${AWS_REGION}" \
                           --no-aggressive-check
EOF

# give access to git user to the directory
sudo chown -R git.git /etc/prereceive

# setup logging directory and give appropriate permissions
sudo mkdir /var/log/prereceive
sudo chown git.git /var/log/prereceive

# create the actuall git hook script by linking to the appropriate directory.
# if the location is wrong for you please consult the appropriate documentation for␣
↪your installation.
sudo mkdir -p /opt/gitlab/embedded/service/gitlab-shell/hooks/pre-receive.d
sudo ln -s /etc/prereceive/pre-receive_active /opt/gitlab/embedded/service/gitlab-
↪shell/hooks/pre-receive.d/pre-receive_active
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

4.1 Submit Feedback

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

4.1.1 Get Started!

Ready to contribute? Here's how to set up *prereceivecli* for local development. Using of pipenv is highly recommended.

1. Clone your fork locally:

```
$ git clone https://github.com/schubergphilis/prereceivecli.git
```

2. Install your local copy into a virtualenv. Assuming you have pipenv installed, this is how you set up your clone for local development:

```
$ cd prereceivecli/
$ pipenv install --ignore-pipfile
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally. Do your development while using the CI capabilities and making sure the code passes lint, test, build and document stages.

4. Commit your changes and push your branch to the server:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

5. Submit a merge request

CHAPTER 5

prereceivecli

5.1 prereceivecli package

5.1.1 Subpackages

prereceivecli.lib package

Submodules

prereceivecli.lib.utils module

Main code for utils.

class prereceivecli.lib.utils.**HashChecker**
Bases: object

Implements a git rebuilding context manager for a pre-receive hook.

verify(*project*, *entries*)
Verifies the protected files or directories specified in the entries.

Parameters

- **project** (Project) – The project object to verify.
- **entries** (*dict*) – The entries of protected files or directories.

Returns A list of errors of the verification failures if any.

Return type errors (list)

class prereceivecli.lib.utils.**Project**(*slug: str*, *group: str*, *git_path: str*, *git_command: str*, *username: str*, *commit: str*, *base: str*)
Bases: object

Models a project exposing attributes for slug, group and git_path.

class prereceivecli.lib.utils.**SecurityEntry**(*hashes: list*, *name: str*, *type: str*)
Bases: object

Models a security entry exposing attributes for slug, type and git_path.

prereceivecli.lib.utils.**execute_command_with_returned_output**(*command*)
Execute the command with returned output.

prereceivecli.lib.utils.**get_project**(*base*, *commit*)
Constructs a project object from a gitlab project path.

Returns An object exposing the required attributes of the environment and the project

Return type (project)

prereceivecli.lib.utils.**get_table_for_project_group**(*project_group*, *credentials*)
Retrieves a dynamodb table following a specific naming convention.

Parameters

- **project_group** (*str*) – The type of the project to look up the table for. Convention states that the table should be named {type, eg:infrastructure}_git_hook.
- **credentials** (AwsCredentials) – An object holding the credentials passed from the authentication process.

Returns if found else None

Return type (dynamodb Table)

prereceivecli.lib.utils.**parse_hook_input**()
Parses the git hook input disregarding the tags reference.

prereceivecli.lib.utils.**send_slack_message**(*webhook*, *message*)
Send a message to a webhook in slack.

Parameters

- **webhook** (*str*) – The webhook to submit the message to
- **message** (*str*) – The message to submit to slack

Returns True on success False otherwise

Return type (bool)

Module contents

lib package.

Import all parts from lib here.

5.1.2 Submodules

5.1.3 prereceivecli.configuration module

Main code for configuration.

5.1.4 prereceivecli.prereceivecli module

Main code for prereceivecli.

class prereceivecli.prereceivecli.**AwsCredentials**(*access_key_id: str*, *secret_access_key: str*, *session_token: str*)

Bases: object

Stores AWS Credentials.

prereceivecli.prereceivecli.**get_arguments**()

Gets us the cli arguments.

Returns the args as parsed from the argsparser.

prereceivecli.prereceivecli.**get_credentials**(*args*)

Gets AWS credentials.

Needs the args to either assume role or get credentials

Credentials: Credentials: The AWS credentials to set for our environment

prereceivecli.prereceivecli.**main**()

Main method.

This method holds what you want to execute when the script is run on command line.

prereceivecli.prereceivecli.**setup_logging**(*level*, *config_file=None*)

Sets up the logging.

Parameters

- **level** – The level to log for.
- **config_file** – The config file with the logging configuration. If provided it superseeds the level arg.

Returns The parsed arguments.

Return type args

prereceivecli.prereceivecli.**validate_commit**(*project*, *dynamodb_table*, *web_hook*, *aggressive_checking*)

Validates that no unauthorized change has been performed on protected files on a specified commit.

Parameters

- **project** (Project) – An object exposing attributes of the required variables.
- **dynamodb_table** (*Table*) – The dynamodb table with the entries for the projects.
- **web_hook** (*str*) – The url of the slack webhook.
- **aggressive_checking** (*bool*) – If set any unmatched repositories will be rejected.

Returns True if the commit is valid False otherwise.

Return type success (bool)

5.1.5 prereceivecli.prereceivecliexceptions module

Custom exception code for prereceivecli.

exception prereceivecli.prereceivecliexceptions.**GitExecutionPathNotFound**
Bases: Exception

Git execution path not found.

5.1.6 Module contents

prereceivecli package.

Import all parts from prereceivecli here.

CHAPTER 6

Credits

6.1 Development Lead

- Costas Tyfoxylos <ctyfoxylos@schubergphilis.com>

6.2 Contributors

- Alberto Rodriguez Garcia <agarcia@schubergphilis.com>
- Ninad Page <npage@schubergphilis.com>

CHAPTER 7

History

CHAPTER 8

0.0.1 (26-02-2019)

- First code creation

CHAPTER 9

0.1.0 (26-02-2019)

- First public release

CHAPTER 10

0.1.1 (29-09-2020)

- Updated quaratine path logic

CHAPTER 11

0.1.2 (02-10-2020)

- Fix for get_project method

CHAPTER 12

0.1.3 (02-10-2020)

- Fixed project_group name logic

CHAPTER 13

0.2.0 (16-10-2020)

- Implemented web identity authentication.

CHAPTER 14

0.2.1 (16-10-2020)

- Bumped dependencies

CHAPTER 15

1.0.0 (30-12-2020)

- Replace hyphens with underscores in all project names for dynamo lookup.

CHAPTER 16

1.0.1 (07-01-2021)

- Made error message friendlier.

CHAPTER 17

1.0.2 (09-03-2021)

- Explicitly targeting the copy git repo for the latest gitlab.

CHAPTER 18

1.1.0 (16-03-2021)

- Implemented archive instead of checkout as checkout corrupted repositories in Gitlab after 13.8 and the exposure of GIT_DIR environment variable.

CHAPTER 19

1.1.1 (22-03-2021)

- Handling special 00000 ref case by not handling it.

CHAPTER 20

1.1.2 (26-04-2021)

- Bumped dependencies.

CHAPTER 21

1.1.3 (08-06-2021)

- Bumped dependencies.

CHAPTER 22

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index