Karr Lab build utilities Documentation

Release 0.0.13

Karr Lab

Aug 19, 2019

Contents

This package performs several aspects of the Karr Lab's build system:

- Versioning with Git and GitHub
  - Creates new Git repositories with the proper directory structure and files for our build system
- Testing code with Python 2 and 3
  - Uses pytest or nose test runners
  - Uses coverage or instrumental for statement, branch, or multiple condition coverage analysis
  - Runs the tests locally or using a Docker image or the CircleCI local executor
- Static code analysis with Pylint
  - Statistically analyzes code using Pylint
- Documentation with Sphinx
  - Generates documentation using Sphinx
- Dependency management
  - Installs and upgrades all of the requirements of a package
  - Identifies missing and unused dependencies
  - Compiles downstream package dependencies
  - Visualizes downstream packages dependencies
  - Checks for cycles in package dependencies
- Continuous integration with CircleCI
  - Creates CircleCI builds for packages
  - Gets, sets, and deletes environment variables
  - Triggers CircleCI to test downstream dependencies
  - Manages passwords used in CircleCI
  - Email notifications
- Test analysis with our test history server
  - Uploads test reports to our test history server
- Coverage analysis with Coveralls
  - Uploads coverage reports to Coveralls
- Coverage analysis and static code analysis with Code Climate
  - Create Code Climate builds for packages
  - Uploads coverage reports to Code Climate
- Distribution with PyPI
  - Uploads packages to PyPI

The build system is primarily designed for:

- Code that is implemented with Python 2/3
- Tests that can be run with pytest
- Code that is documented with Sphinx in Napolean/Google style

- Code that is versioned with Git/GitHub
- Builds that are run on CircleCI
- Coverage reports that are hosted on Coveralls and Code Climate
- Documentation that is hosted on Read the Docs

CHAPTER 1

Contents

1.1 Installation

1.1.1 Pre-requisites

1. Run these commands to install the required packages on Ubuntu:

```
# install OS packages
apt-get update
apt-get install \
    cmake \
    enchant \
    gcc \
    git \
    graphviz \
    openssh-client \
    pandoc \
    python \
    python-pip \
    wget

# install libgit2 (version in apt repository is old)
pushd /tmp
wget https://github.com/libgit2/libgit2/archive/v0.26.3.tar.gz -O /tmp/libgit2-0.
↪26.3.tar.gz
tar -xvvf /tmp/libgit2-0.26.3.tar.gz
cd /tmp/libgit2-0.26.3
cmake .
make
make install
ldconfig
cd /tmp
export LIBGIT2=/usr/local
echo "" >> ~/.bashrc
```

(continues on next page)

(continued from previous page)

```
echo "# libgit2" >> ~/.bashrc
echo "export LIBGIT2=/usr/local" >> ~/.bashrc
source ~/.bashrc
rm /tmp/libgit2-0.26.3.tar.gz
rm -r /tmp/libgit2-0.26.3
popd
```

2. Run this command to upgrade pip and setuptools:

```
pip install -U pip setuptools
```

3. Optionally, create `~/.wc/karr_lab_build_utils.cfg` to configure the package (see configuration options in `karr_lab_build_utils/config/core.schema.cfg`). We recommend that Karr Lab members start by copying the shared configuration file from `karr_lab_build_config/karr_lab_build_utils.cfg`.
4. Optionally, create an SSH key for GitHub. This is needed to run tests using Docker and CircleCI.
    1. Create an SSH key and save it to `~/.ssh/id_rsa`:

    ```
    ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
    eval $(ssh-agent -s)
    ssh-add ~/.ssh/id_rsa
    ```

    2. Copy the contents of `~/.ssh/id_rsa.pub` and use it to add your SSH key to GitHub following the instructions at https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account.
    3. Configure git to use SSH by saving the following to `~/.gitconfig`:

    ```
    [url "ssh://git@github.com/"]
        insteadOf = https://github.com/
    ```

5. Optionally install Docker by following the installation instructions in "An Introduction to Whole-Cell Modeling." This is needed to run tests using Docker and CircleCI.
6. Optionally, install the CircleCI command line tool by following the installation instructions in "An Introduction to Whole-Cell Modeling." This is needed to run tests using CircleCI.
7. Optionally, create a PyPI account at https://pypi.python.org. This is needed to upload packages to PyPI.
8. Optionally, save your PyPI credentials to `~/.pypirc`. This is needed to upload packages to PyPI.:

```
[distutils]
index-servers =
    pypi

[pypi]
repository: https://upload.pypi.org/legacy/
username: <username>
password: <password>
```

1.1.2 Install the latest revision from GitHub

Run the following command to install the latest version from GitHub:

```
pip install git+https://github.com/KarrLab/sphinxcontrib-googleanalytics.git
↪#egg=sphinxcontrib_googleanalytics
pip install git+https://github.com/KarrLab/pkg_utils.git#egg=pkg_utils
pip install git+https://github.com/KarrLab/wc_utils.git#egg=wc_utils
pip install git+https://github.com/KarrLab/karr_lab_build_utils.git#egg=karr_lab_
↪build_utils
```

1.2 Tutorial for WC modeling software developers

1.2.1 Creating a new package

Run this command to create a new package (create local and remote repositories with the proper directory structure and files for our build system, add repository to CircleCI, add package to downstream dependencies of dependencies, etc.). The command will prompt you for all of the information needed to create a repository and instruct you how to create a new package, including linking it to CircleCI, Coveralls, Code Climate, and Read the Docs. The command should be run from the package's desired parent directory, e.g. with a current working directory of `~/Documents`.:

```
cd ~/Documents
karr_lab_build_utils create-package
```

`karr_lab_build_utils` also provides two lower-level commands for creating, cloning, and initializing Git repositories. These commands are an alternative to the `create-package` command which creates, clones, and initializes Git repositories and much more.

- `create-repository`: create a new GitHub repository and clone it locally
- `setup-repository`: set up the file structure of a local Git repository

```
cd ~/Documents
karr_lab_build_utils create-repository
karr_lab_build_utils setup-repository
```

These commands will create a repository with the following directory structure and files:

```
/path/to/repo/
LICENSE
setup.py
setup.cfg
MANIFEST.in
requirements.txt
requirements.optional.txt
README.md
.karr_lab_build_utils.yml
.gitignore
<repo_name>
    __init__.py
    VERSION
    __main__.py (optional, for command line programs)
tests/
    requirements.txt
    fixtures/
docs/
    conf.py
    requirements.txt
```

(continues on next page)

(continued from previous page)

```
    requirements.rtd.txt
    index.rst
```

1.2.2 Developing a package

Please see the Software engineering section of "An Introduction to Whole-Cell Modeling."

1.2.3 Managing dependencies of packages

Installing the dependencies for a package

Run the following command to install all of the requirements for the current package in the following files:

- `requirements.txt`,
- `requirements.optional.txt`,
- `tests/requirements.txt`, and
- `docs/requirements.txt`

```
karr_lab_build_utils install-requirements
```

Finding missing requirements for a package

Run this command to find potentially missing requirements for a package:

```
karr_lab_build_utils find-missing-requirements
```

Finding unused requirements for a package

Run this command to identify potentially unused requirements for a package:

```
karr_lab_build_utils find-unused-requirements
```

Compiling the downstream dependencies of a package

1. Clone all of our packages
2. Run this command to compile the downstream dependencies of your package:

   ```
   karr_lab_build_utils compile-downstream-dependencies --packages-parent-dir ~/
   ↪Documents
   ```

3. Optionally, add the `--downstream-dependencies-filename` option to save the dependencies to a YAML file:

   ```
   karr_lab_build_utils compile-downstream-dependencies --packages-parent-dir ~/
   ↪Documents --downstream-dependencies-filename .circleci/downstream_dependencies.
   ↪yml
   ```

1.2.4 Configuring packages

The `karr_lab_build_config` repository should contain all of the whole-cell modeling and third party configuration files needed to run your tests. This should include all usernames, passwords, and tokens needed to run your tests.

Configuration files for whole-cell modeling software should be saved to the top-level directory of the `karr_lab_build_config` repository with the file pattern `<package_name>.cfg`.

All configuration files for third-party software should be saved to the `third_party` subdirectory of the `karr_lab_build_config` repository. In addition, `third_party/paths.yml` should contain a YAML-formatted dictionary whose keys are the names of the files in the `third_party` subdirectory and whose values are the locations that these files should be copied to.

1.2.5 Testing with pytest, coverage, instrumental, Docker, and CircleCI

Running the tests for a package

Run this command to test the local package:

```
karr_lab_build_utils run-tests
```

Evaluating the coverage of the tests

Add the `--coverage-type` option to specify `statement`, `branch`, or `multiple-condition` coverage, e.g.:

```
karr_lab_build_utils run-tests --with-coverage --coverage-type branch
```

Running tests with Docker or the CircleCI local executor

Add the `--environment` option to specify `local`, `docker`, or `circleci`, e.g.:

```
karr_lab_build_utils run-tests --environment docker tests
```

1.2.6 Configuring tests of downstream dependencies

The `downstream_dependencies` key of `/path/to/repo/.karr_lab_build_utils.yml` should represent a list of the names of the downstream dependencies of your package. For example, if your package is used by `wc_lang` and `wc_sim`, `.karr_lab_build_utils.yml` should contain:

```
downstream_dependencies:
  - wc_lang
  - wc_sim
```

1.2.7 Configuring the static analyses run by the build system

The `static_analyses.ignore_files` key of `/path/to/repo/.karr_lab_build_utils.yml` should represent a list of glob patterns not to statically analyze. E.g.:

```
static_analyses:
  ignore_files:
      - karr_lab_build_utils/templates/*
```

1.2.8 Configuring build email notifications

The `email_notifications` key of `/path/to/repo/.karr_lab_build_utils.yml` should represent a list of email addresses to receive notifications of the build status of your package. E.g.:

```
email_notifications:
  - jonrkarr@gmail.com
```

1.2.9 Documenting code with Sphinx

Building the documentation for a package

Run this command to compile the documentation in HTML format for a package.:

```
karr_lab_build_utils make-documentation
```

Spell checking documentation

Add the `--spell-check` option to spell check the documentation, e.g.:

```
karr_lab_build_utils -spell-check make-documentation
```

The output will be saved to `docs/_build/spelling/output.txt`.

White-listed words can be saved (1 word per line) to `docs/spelling_wordlist.txt`.

1.3 Tutorial for build administrators

The following is a brief tutorial of the command line interface for `karr_lab_build_utils`. Note, the command line interface provides some functionality in addition to that described below. However, in general, these additional commands should only be run from CircleCI.

Except as indicated below, `karr_lab_build_utils` should be run from the package's root directory, e.g. with a current working directory of `~/Documents/my_package`.

To use the command line interface, your package should follow the organization scheme described in "An Introduction Whole-Cell Modeling":

- Structuring Python projects
- Testing Python projects
- Documenting Python code
- Packaging Python projects

1.3.1 Getting help

Run the following commands to get help documentation about the command line utility and each individual command:

```
karr_lab_build_utils --help
karr_lab_build_utils create-repository --help
```

1.3.2 Versioning with Git and GitHub

Creating a repository for a package

Run this command to create a new repository (including both local and GitHub versions). This should be run from the package's desired parent directory, e.g. with a current working directory of `~/Documents`.:

```
cd ~/Documents
karr_lab_build_utils create-repository repository_name \
    --description description \
    --public
```

1.3.3 Statically analyzing code with Pylint

Statically analyzing a package with Pylint

Run this command to statically analyze a package using **'Pylint https://www.pylint.org/'_**:

```
karr_lab_build_utils analyze-package package_name
```

This will identify potential errors such as

- duplicate arguments
- duplicate dictionary keys
- re-imported modules, classes, functions, and variables
- unused imports, arguments, and variables
- wild card imports

Visualizing all of the package dependencies

1. Clone all of our packages
2. Run this command to visualize the dependencies of your packages:

```
karr_lab_build_utils visualize-package-dependencies --packages-parent-dir ~/
↪Documents --out-filename ~/Documents/package-dependencies.pdf
```

1.3.4 Continuous integration with CircleCI

The commands described in this section require a CircleCI API token. Visit https://circleci.com/account/api to create a token.

Following a build for a package

Run this command to follow a CircleCI build for a package instead of using the CirlceCI web interface:

```
karr_lab_build_utils follow-circleci-build \
    --repo-owner <repo_owner> \
    --repo-name <repo_name>
```

Getting the environment variables for a package

Run this command to get the CircleCI environment variables for a package:

```
karr_lab_build_utils get-circleci-environment-variables \
    --repo-owner <repo_owner> \
    --repo-name <repo_name>
```

Setting a environment variable for a package

Run this command to set a CircleCI environment variable for a package:

```
karr_lab_build_utils set-circleci-environment-variable <name> <value> \
    --repo-owner <repo_owner> \
    --repo-name <repo_name>
```

Deleting a environment variable for a package

Run this command to delete a CircleCI environment variable for a package:

```
karr_lab_build_utils delete-circleci-environment-variable <name> \
    --repo-owner <repo_owner> \
    --repo-name <repo_name>
```

Triggering testing downstream dependencies of a package

1. Save a list of your the downstream dependencies of the package in YAML format to `.circleci/downstream_dependencies.yml`, e.g.:

   ```
   - wc_lang
   - wc_sim
   ```

2. Run this command to trigger CircleCI to test the downstream dependencies of your package:

   ```
   karr_lab_build_utils trigger-tests-of-downstream-dependencies
   ```

1.3.5 Statically analyzing code and performing coverage analysis with Code Climate

Creating a Code Climate build for a package

Run this command to create a Code Climate build for a package instead of using the Code Climate web interface:

```
karr_lab_build_utils create-codeclimate-github-webhook \
    --repo-owner <repo_owner> \
    --repo-name <repo_name>
```

1.3.6 Distributing packages with PyPI

Distributing a package by uploading it to PyPI

1. Create a PyPI account
2. Save your credentials to ~/.pypirc:

```
[distutils]
index-servers =
    pypi

[pypi]
repository: https://upload.pypi.org/legacy/
username: <username>
password: <password>
```

3. Run this command to upload your package to PyPI:

```
karr_lab_build_utils upload-package-to-pypi
```

1.4 About

1.4.1 License

The software is released under the MIT license

```
The MIT License (MIT)

Copyright (c) 2016 Karr Lab

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.4.2 Development team

This package was developed by the Karr Lab at the Icahn School of Medicine at Mount Sinai in New York, USA.

1.4.3 Acknowledgments

This work was supported by a National Institute of Health MIRA award [grant number 1 R35 GM 119771-01]; a National Science Foundation INSPIRE award [grant number 1649014]; and the National Science Foundation / ERASynBio [grant numbers 1548123, 335672].

1.4.4 Questions and comments

Please contact the Karr Lab with any questions or comments.