

Sub Org Info

coala

Student Info

Name: Karan Sharma

Alternate names: mr-karan

Email: karansharma1295@gmail.com

Telephone: +91 9650 31 8721

GSoC Blog RSS Feed URL: <http://gsoc2016mr-karan.blogspot.com/feeds/posts/default>

Timezone: IST (UTC +5:30)

CV: [Link to CV](#)

Github: [mr-karan](#)

Code Sample

- [coala](#)
- [coala-bears](#)

Prominent PR's from the above:

1. [SCSSLintBear](#)
2. [BootLintBear](#)
3. [formatRLintBear](#)
4. [RuboCopBear](#)
5. [AutoPrefixBear](#)
6. [GoTypeBear](#)

Project Title: Extend linter integration

Proposal Abstract

The aim of the project is to ease the process of creating a new Lint Bear, address the issue of linting files having embedded source code and provide command line interface improvements to existing coala application.

Possible Mentors : @sils1297, @Abdealijk, @Makman2

Languages used : Python

The Problem and Motivation

(A) coala-bears --create command

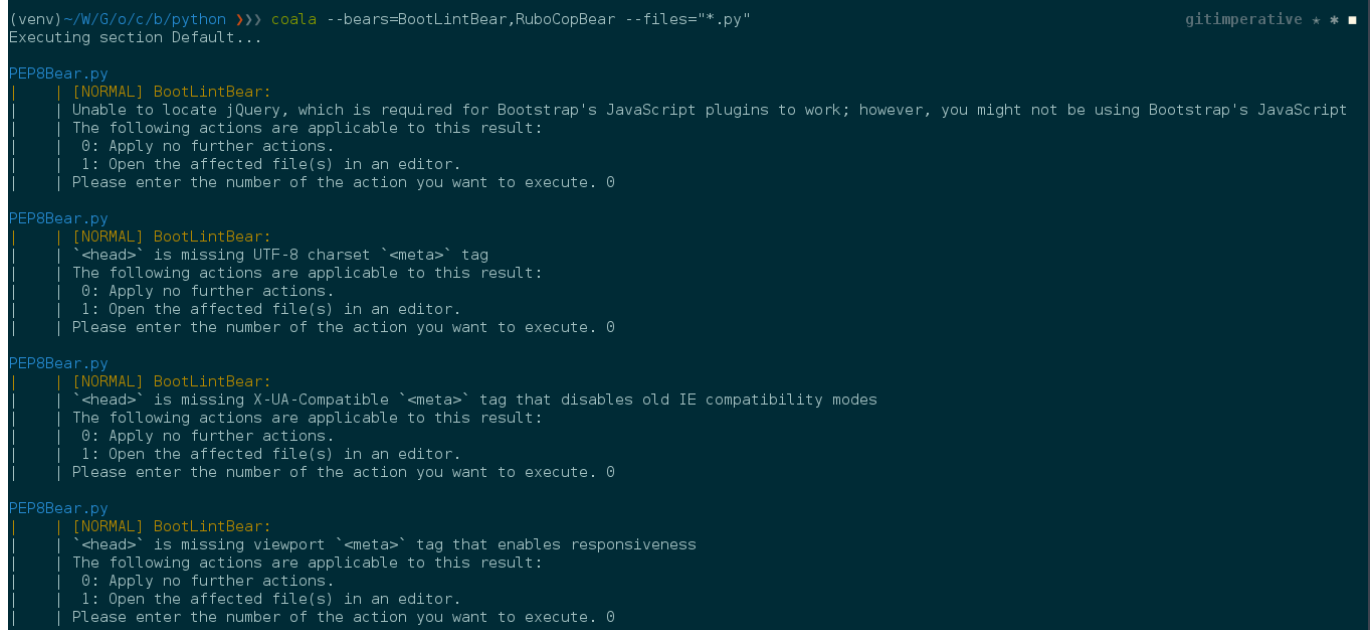
The current process of creating a new bear is quite repetitive and mundane. Each bear which has been added to coala has to go through these 2 steps:

- Manually add dependencies to `package.json/requirements.txt/Gemfile` if it's based on either Node/ Python/ Ruby.
- Create files in directories `Bears` and `Tests`.

This motivates me strongly to integrate this command with coala and make the process of creating a new bear easier. I've also filed an issue regarding the same [here](#).

(B) Multi Syntax Lint Bear

Another current problem which is presented in [issue #1690](#) is, handling source code files having embedded syntax. Elaborating more on it, currently the user has to run language specific linters for files which can be quite a cumbersome process if the directory has files of different programming language syntaxes. Another problem is that the linters don't check for the extension of the file they are linting. Case in point, the following screenshot shows `Bootstrap` and `Ruby Lint` Bears linting a `python` file.



```
(venv)~/W/G/o/c/b/python >>> coala --bears=BootLintBear,RuboCopBear --files="*.py"
Executing section Default...

PEP8Bear.py
| [NORMAL] BootLintBear:
| Unable to locate jQuery, which is required for Bootstrap's JavaScript plugins to work; however, you might not be using Bootstrap's JavaScript
| The following actions are applicable to this result:
| 0: Apply no further actions.
| 1: Open the affected file(s) in an editor.
| Please enter the number of the action you want to execute. 0

PEP8Bear.py
| [NORMAL] BootLintBear:
| `<head>` is missing UTF-8 charset `<meta>` tag
| The following actions are applicable to this result:
| 0: Apply no further actions.
| 1: Open the affected file(s) in an editor.
| Please enter the number of the action you want to execute. 0

PEP8Bear.py
| [NORMAL] BootLintBear:
| `<head>` is missing X-UA-Compatible `<meta>` tag that disables old IE compatibility modes
| The following actions are applicable to this result:
| 0: Apply no further actions.
| 1: Open the affected file(s) in an editor.
| Please enter the number of the action you want to execute. 0

PEP8Bear.py
| [NORMAL] BootLintBear:
| `<head>` is missing viewport `<meta>` tag that enables responsiveness
| The following actions are applicable to this result:
| 0: Apply no further actions.
| 1: Open the affected file(s) in an editor.
| Please enter the number of the action you want to execute. 0
```

(C) Navigation of errors

Another feature that can be implemented in the `ConsoleInteraction` is the ability to navigate the errors/warnings shown by the bear to the user. Currently if the user chooses to go over the next warning, he/she cannot come back to the previous error. While the error/warning is still displayed on the screen but if the lint has an option for auto correct and the user mistakenly chose to go over the next error then he/she has to run the bear again to correct that line.

(D) Multiline regex support

Currently there's a workaround for the linter executable which outputs multiline message, i.e. by

using regex hacks. A new `Lint` class variable should be introduced to provide a cleaner solution.

(E) Improving the UI of coala

Since coala is a CLI application, I have looked at various modules and found [python prompt toolkit](#) to be a versatile library which can help in improving command line interface.

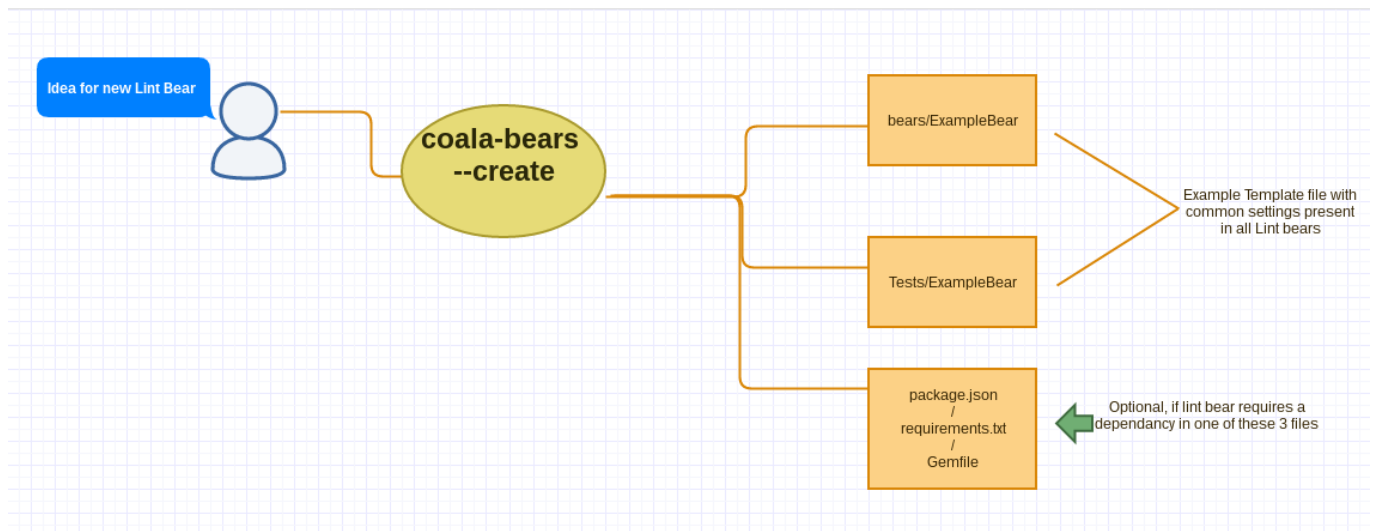
(F) A Web application for listing out all Bears (optional)

An optional feature (if time permits) that i would like to add in my proposal is the creation of a small web application which will host details of all the bears developed till now. This project will complement the [Decentralizing Bear](#) project. Since every bear is going to be split into it's own python package and Github repositories, I think having a central place for details about all Bears developed till now with their help guides/documentation will be a good idea. Inspiration of this project is from [Atom linter list](#) website and [postcss](#).

Implementation

Phase 1: Designing the approach for coala-bears --create command

A separate repository will be created for `coala-bears --create` executable. The user will be greeted with a small set of questionnaire in the CLI which will initiate the bears and tests folders and also add external linter executable dependencies. An example template and a questionnaire will be created in `JSON` file.



The approach of this command will be to work like how `git --interactive` works or [pelican-quickstart](#) works. The questionnaire will have the following design:

```

~ >>> >>coala-bears --create
Welcome to coala-bears! Enter the name of your bear.
>> ExampleLintBear
Great! bears/ExampleLintBear and tests/ExampleLintBear has been initialized.
Which language is the Lint executable based on? (Ruby/Python/Js/Other)
>>Python
Enter the name of your lint executable
>>flake8
Great! flake8 has been added to requirements.txt
Congrats! Now you may start modifying the default template and add tests for your new bear.

```

Generated files from this command will have basic syntax which is required for bear to run:

ExampleLintBear.py

```

import re

from coalib.bearlib.abstractions.Lint import Lint
from coalib.bears.LocalBear import LocalBear
from coalib.results.RESULT_SEVERITY import RESULT_SEVERITY

class ExampleBear(LocalBear, Lint):
    executable = 'user_input_binary'
    output_regex = re.compile(r'')
    severity_map = {
        "": RESULT_SEVERITY.NORMAL,
        "": RESULT_SEVERITY.MAJOR,
        "": RESULT_SEVERITY.INFO
    }

    def run(self, filename, file):
        '''
        Checks the code with ``user_input_binary`` on each file separately.
        '''
        return self.lint(filename)

```

ExampleLintBearTest.py

```

from bears.examplebeardir.ExampleLintBear import ExampleLintBear
from tests.LocalBearTestHelper import verify_local_bear

good_file = """
""".splitlines(keepends=True)

bad_file = """
""".splitlines(keepends=True)

ExampleLintBearTest = verify_local_bear(ExampleLintBear,
                                         valid_files=(good_file,),
                                         invalid_files=(bad_file,))

```

Progress on this work during Application period can be seen at:

<https://github.com/coala-analyzer/coala-bears/issues/154#issuecomment-192676559>

Phase 2: Working with Lint Class

In this phase, I will be working on adding a few `Lint` class methods and variables. I'll start by working on implementing a Multi Syntax Lint Bear and will be adding `can_lint()` method which will assert whether the bear is meant to lint that syntax or not. If `can_lint()` returns `True`, program execution will follow normal routine but if it returns `False` then it will find the list of Bears which can lint that syntax. If the source code has embedded source code in it (like `HTML` files may have `.php`, `.css` syntax in it as well) a new `Lint` class variable `syntax` is introduced for this purpose. The language syntax will be mapped to the portion of code which is separate from original syntax which can be extracted using `Regex`. An example of this is linting a `HTML` file which has `JS` and `CSS` code embedded in it.

```
embedded_syntax = ('css','js')
embedded_syntax_regex = {'css':
#regex for detecting css code in <style></style> tags in html;
'Js': # regex for detecting js code in <script></script> tags
}
```

So, the code extracted from these variables will be linted using appropriate bears. The user can define which bear to run in `.coafile` or we can display all the bears present for that syntax to the user and the user can choose during run time.

Also, a new `multiline` `Lint` class variable will be introduced which will use `re.finditer` module to iterate over regex output for lint executables using multiline output messages. For navigation of results, two methods `get_next()` and `get_previous()` will be implemented in `ConsoleInteraction` class to navigate the results shown to the user.

Phase 3: Working on improving the command line interface

For improving the current command line interface, I will be implementing the following features from Python Prompt Toolkit Library:

1. [Pygment syntax highlighting](#)
2. [Autocompletion](#)
3. [Status Bar](#)

For syntax-highlighting, I've filed an issue at [coala](#)

Phase 4: Web application for listing out all Bears (Optional)

I will begin my work by creating a small web application in Flask. I will write a small program which will parse the list of bears like `coala -A` command does and will plug in them to the website. If the bears have a small documentation written in Markdown files, I will add them to the details page of each bear. A script can also be written which will write the details of each bear to a `JSON` file. The fields of `JSON` can be This will make the task of creating a web application very easy process by just plugging in the values to the template.

```
{
  name: " ",
  Ascinema: " " # ascinema url of working of bear,
  Short-description: " ",
  Long-description: " ":
  Author Twitter/Github handle: " ",
}
```

The front end of website can be done in any lightweight framework like Polymer.

Timeline(Tentative)

Community Bonding Period (22 April - 22 May)

Goal: Community Bonding

- The principal focus in this period will be studying in depth about coala, diving deep in the codebase, the functionalities of coala, making notes as I go along.
- I'll ask guidance from my mentor upon the functioning of the `Multi Syntax Bear` because that is the most crucial part of my project.
- If possible, I plan to start coding in this period only, so that I get a head-start.

Week 1 (23 May - 30 May)

Goal: Creating the coala-bears --create helper functions

- I'll start with making a questionnaire in `JSON` format.
- I'll write helper functions for generating files based on user input.
- I'll write helper function for appending linter executable in external dependency files.

Week 2 (30 May - 6 June)

Goal: Completing coala-bears command

- I'll integrate the helper functions in coala-bears module.
- I'll write test cases for `coala-bears --create` command covering boundary conditions and handling unexpected user behavior.
- By the end of Week 2, `coala-bears --create` will be completed.

Week 3 (6 June - 13 June)

Goal: Implementing multi syntax Lint bear

- I'll start first with implementing `can_lint` method in Lint class. I'll take code base of [Sublime Linter Plugin](#) as my starting point.
- I'll be working on linting the bear to work on only the files it's supposed to lint using `can_lint()` method and displaying appropriate error/ other Lint Bear choices to the user.

Week 4 (13 June-20 June)

Goal: Implementing embedded source code checking

- I'll be working on handling the Lint method for embedded source code.
- I'll be writing test cases for handling embedded source code.

Week 5 (20 June - 27 June)

Goal: Mid term Evaluation

- Have a fully functional `coala-bears --create` command and implemented `Multi Syntax Lint Bear`.
- Take feedback from mentors and improve upon it.
- Fix bugs if found any/reported.

Week 6 - 7 (27 June - 11 July)

Goal: Working on multiline regex and navigation of results

- I will be working on multiline regex support.
- I will be working on Navigation of Results.
- I will be writing test cases for `get_next` and `get_previous` and will test multiline with Linter such as `write-good` which outputs multiline message.

Week 8 - Week 9 (11 July - 25 July)

Goal: Modifying interface of coala CLI application

- I will be working on [Issue #1925](#).
- I will be adding the rest of features mentioned above from Python Prompt Toolkit Library to coala CLI.

Week 10 (25 July - 1st August)

Goal: Buffer Week

- In this period I'll pause the current development of my project and will be doing bug fixing, working on mentor's feedback and finish incomplete tasks if the above mentioned schedule didn't work out due to unforeseen circumstances.

Week 11 - Week 12 (1st August - 15 August)

Goal: Working with web application.

- Create a web application using `Flask` which will have a list of all bears present with a searchable catalog.
- I will be hosting this web application on `coala-html` repository.

Week 13 (15 August - 23 August))

Goal: Buffer Week; Finishing all pending tasks

- I will be documenting all the changes.
- I will be tidying up the code and refactoring if deemed necessary.
- I will do additional testing and bug fixing of all the changes done till now.

Future Work:

- I would love to maintain `coala-bears --create` command for further improvements and also the coala bears website.
 - I will be writing blog posts regularly about working on coala project and will share whatever I learn from this project in my blog.
 - I will be encouraging new comers on helping how to create a new Lint Bear.
-

Other Commitments

- Do you have any other commitments during the main GSoC time period, May 23rd to August 23rd?
 - None, I'll be having my summer break in this period.
- Do you have exams or classes that overlap with this period?
 - I don't have any exams though my college will start from August 1 but since it will be a new semester there won't be any work load or exams coming up till August 23.
- Do you plan to have any other jobs or internships during this period?
 - No I'll be working full time for GSoC project.
- Do you have any other short term commitments during this period?
 - None
- Have you applied with any other organizations? If so, do you have a preferred project/org? (This will help us in the event that more than one organization decides they wish to accept your proposal.)
 - No. coala is my first and only preference.

extra information

- Will you be able to join us for a conference such as EuroPython or GUADEC and present your work if at least partial sponsorship will be provided? (See <https://github.com/coala-analyzer/coala/wiki/Conferences-Upcoming>)
 - Yes, in fact I have registered for EuroPython already and wish to give a poster presentation over there. I have also applied for giving a workshop at PyCon India. [Here's](#) my proposal for the same.
- We love having twitter handles so we can tell people publicly about your great project and successes at https://twitter.com/coala_analyzer! (Not required but recommended.
 - A) [karansharma1295](#)