Language Savant. If your repository's language is being reported incorrectly, send us a pull request!

README.md

Linguist

We use this library at GitHub to detect blob languages, highlight code, ignore binary files, suppress generated files in diffs, and generate language breakdown graphs.

Features

Language detection

Linguist defines a list of all languages known to GitHub in a yaml file. In order for a file to be highlighted, a language and a lexer must be defined there.

Most languages are detected by their file extension. For disambiguating between files with common extensions, we first apply some common-sense heuristics to pick out obvious languages. After that, we use a statistical classifier. This process can help us tell the difference between, for example, `.h` files which could either be C, C++, or Obj-C.

```
Linguist::FileBlob.new("lib/linguist.rb").language.name #=> "Ruby"

Linguist::FileBlob.new("bin/linguist").language.name #=> "Ruby"
```

See lib/linguist/language.rb and lib/linguist/languages.yml.

Syntax Highlighting

The actual syntax highlighting is handled by our Pygments wrapper, pygments.rb. It also provides a Lexer abstraction that determines which highlighter should be used on a file.

Stats

The Language stats bar that you see on every repository is built by aggregating the languages of each file in that repository. The top language in the graph determines the project's primary language.

The repository stats API, accessed through `#languages`, can be used on a directory:

API UPDATE

Since Version 3.0.0 Linguist expects a git repository (in the form of a Rugged::Repository) to be passed when initializing `Linguist::Repository`.

```
require 'rugged'
require 'linguist'

repo = Rugged::Repository.new('.')
project = Linguist::Repository.new(repo, repo.head.target_id)
project.language     #=> "Ruby"
project.languages    #=> { "Ruby" => 119387 }
```

These stats are also printed out by the `linguist` binary. You can use the `--breakdown` flag, and the binary will also output the breakdown of files by language.

You can try running `linguist` on the root directory in this repository itself:

```
$ bundle exec linguist --breakdown

100.00% Ruby

Ruby:
Gemfile
Rakefile
bin/linguist
github-linguist.gemspec
lib/linguist.rb
lib/linguist/blob_helper.rb
lib/linguist/classifier.rb
lib/linguist/file_blob.rb
lib/linguist/generated.rb
lib/linguist/heuristics.rb
lib/linguist/language.rb
lib/linguist/lazy_blob.rb
lib/linguist/md5.rb
lib/linguist/repository.rb
lib/linguist/samples.rb
lib/linguist/tokenizer.rb
lib/linguist/version.rb
test/test_blob.rb
test/test_classifier.rb
test/test_heuristics.rb
test/test_language.rb
test/test_md5.rb
test/test_pedantic.rb
test/test_repository.rb
test/test_samples.rb
test/test_tokenizer.rb
```

Ignore vendored files

Checking other code into your git repo is a common practice. But this often inflates your project's language stats and may even cause your project to be labeled as another language. We are able to identify many of these files and directories and exclude them.

```
Linguist::FileBlob.new("vendor/plugins/foo.rb").vendored? # => true
```

See Linguist::BlobHelper#vendored? and lib/linguist/vendor.yml.

Generated file detection

Not all plain text files are true source files. Generated files like minified js and compiled CoffeeScript can be detected and excluded from language stats. As an extra bonus, these files are suppressed in diffs.

```
Linguist::FileBlob.new("underscore.min.js").generated? # => true
```

See Linguist::Generated#generated?.

Overrides

Linguist supports custom overrides for language definitions and vendored paths. Add a `.gitattributes` file to your project using the keys `linguist-language` and `linguist-vendored` with the standard git-style path matchers for the files you want to override.

```
$ cat .gitattributes
*.rb linguist-language=Java

$ linguist --breakdown
100.00% Java

Java:
ruby_file.rb
```

By default, Linguist treats all of the paths defined in lib/linguist/vendor.yml as vendored and therefore doesn't include them in the language statistics for a repository. Use the `linguist-vendored` attribute to vendor or un-vendor paths.

```
$ cat .gitattributes
special-vendored-path/* linguist-vendored
jquery.js linguist-vendored=false
```

Installation

Github.com is usually running the latest version of the `github-linguist` gem that is released on RubyGems.org.

But for development you are going to want to checkout out the source. To get it, clone the repo and run Bundler to install its dependencies.

```
git clone https://github.com/github/linguist.git
cd linguist/
bundle install
```

To run the tests:

```
bundle exec rake test
```

Contributing

The majority of contributions won't need to touch any Ruby code at all. The master language list is just a YAML configuration file.

We try to only add languages once they have some usage on GitHub, so please note in-the-wild usage examples in your pull request.

Almost all bug fixes or new language additions should come with some additional code samples. Just drop them under `samples/` in the correct subdirectory and our test suite will automatically test them. In most cases you shouldn't need to add any new assertions.

A note on language extensions

Linguist has a number of methods available to it for identifying the language of a particular file. The initial lookup is based upon the extension of the file, possible file extensions are defined in an array called `extensions`. Take a look at this example for example for `Perl`:

```
Perl:
  type: programming
  ace_mode: perl
  color: "#0298c3"
  extensions:
  - .pl
  - .PL
  - .perl
  - .ph
  - .plx
  - .pm
  - .pod
  - .psgi
  interpreters:
  - perl
```

Any of the extensions defined are valid but the first in this array should be the most popular.

Testing

Sometimes getting the tests running can be too much work, especially if you don't have much Ruby experience. It's okay: be lazy and let our build bot Travis run the tests for you. Just open a pull request and the bot will start cranking away.

Here's our current build status, which is hopefully green: build passing

Releasing

If you are the current maintainer of this gem:

1. Create a branch for the release: `git checkout -b cut-release-vxx.xx.xx`
2. Make sure your local dependencies are up to date: `bundle install`
3. Ensure that samples are updated: `bundle exec rake samples`
4. Ensure that tests are green: `bundle exec rake test`
5. Bump gem version in `lib/linguist/version.rb`. For example, like this.
6. Make a PR to github/linguist. For example, #1238.
7. Build a local gem: `bundle exec rake build_gem`
8. Testing:
   i. Bump the Gemfile and Gemfile.lock versions for an app which relies on this gem
   ii. Install the new gem locally
   iii. Test behavior locally, branch deploy, whatever needs to happen
9. Merge github/linguist PR
10. Tag and push: `git tag vx.xx.xx; git push --tags`
11. Push to rubygems.org -- `gem push github-linguist-3.0.0.gem`