

Ruby Rules™ for Bazel

Yuki (Yugui) Sonoda, Konstantin Gredeskoul & Contributors.

Table of Contents

1. Build Status & Activity	2
2. Rules Development Status	3
3. Table of Contents	4
4. Usage	5
4.1. <code>WORKSPACE</code> File	5
4.1.1. Load dependencies, select Ruby SDK and define one or more Bundles	5
4.2. <code>BUILD.bazel</code> file(s)	6
4.2.1. Define Ruby Executable, Library and an RSpec	6
4.2.2. Package Ruby files as a Gem	7
4.3. Tool Specific Setup	8
4.3.1. ASDF	8
4.4. Rule Dependency Diagram	9
5. Rules	10
5.1. <code>ruby_library</code>	10
5.2. <code>ruby_binary</code>	11
5.3. <code>ruby_test</code>	12
5.4. <code>ruby_bundle</code>	13
5.4.1. Conventions	15
5.4.2. <code>WORKSPACE</code> : Vendor directory handling	15
5.4.3. <code>BUILD.bazel</code> :	15
5.5. <code>ruby_rspec</code>	16
5.6. <code>ruby_gem</code>	17
6. Potential Future Features	20
7. Contributing	21
7.1. Setup	21
7.1.1. Using the Script	21
7.1.2. OS-Specific Setup	22
7.2. Verifying Your Environment	22
7.2.1. Issues During Setup	22
7.3. Developing Rules	22
7.4. Running Tests	23
7.4.1. Test Script	23
7.5. Linter	23
7.6. Regenerating README.pdf & Changelog	24
8. Copyright	25

This repo is primarily maintained by [Konstantin Gredeskoul](#) and [Yuki "Yugui" Sonoda](#). We are both very busy and would really love more contributors to join the core team. If you are interested in developing Ruby Rules for Bazel, please submit a couple of PRs and then lets talk!

!

You can read or print this README in a proper PDF format by grabbing our [README.pdf](#).

Chapter 1. Build Status & Activity

CI Status	Activity & Documentation
[CircleCI] Ê	[activity] Ê
[Build Status] Ê	<img src="/var/folders/jq/853fg3814rs6xx_zxk9sgsv40000gn/T/image-20211109- 66422-mtnk4u" format="" alt="changelog" width="0"> <img src="/var/folders/jq/853fg3814rs6xx_zxk9sgsv40000gn/T/image-20211109- 66422-6q8jtw" format="" alt="readme.pdf" width="0">

Chapter 2. Rules Development Status

Readiness	Types of Applications
Development Status Ready	ruby apps, ruby gems, micro-services, ideally in a mono-repo
Development Status Ready	medium-sized Ruby on Rails apps, ideally in a mono-repo
Development Status Wait	complex Ruby on Rails monoliths, single-repo

!

we have a short guide on [Building your first Ruby Project](#) on the Wiki. We encourage you to check it out.

Chapter 3. Table of Contents

¥ Ruby Rules™ for Bazel Build System

- ! Build Status & Activity

- ! Rules Development Status

- ! Table of Contents

- ! Usage

 - " WORKSPACE File

 - " BUILD.bazel file(s) * Tool Specific Setup * Rule Dependency Diagram

- ! Rules

 - " ruby_library

 - " ruby_binary

 - " ruby_test

 - " ruby_bundle

 - " ruby_rspec

 - " ruby_gem

- ! Potential Future Features

- ! Contributing

 - " Setup

 - " Verifying Your Environment

 - " Developing Rules

 - " Running Tests

 - " Linter

 - " Regenerating README.pdf & Changelog

- ! Copyright

Chapter 4. Usage

4.1. WORKSPACE File

4.1.1. Load dependencies, select Ruby SDK and define one or more Bundles

```
workspace(name = "my_ruby_project")

load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")
load("@bazel_tools//tools/build_defs/repo:git.bzl", "git_repository")

#####
# To get the latest ruby rules, grab the 'master' branch.
#####

git_repository(
    name = "bazel_ruby_rules_ruby",
    remote = "https://github.com/bazel-ruby/rules_ruby.git",
    branch = "master"
)

load(
    "@bazel_ruby_rules_ruby//ruby:deps.bzl",
    "rules_ruby_dependencies",
    "rules_ruby_select_sdk",
)

rules_ruby_dependencies()

#####
# Specify Ruby version. This will either build Ruby or use a local
# RENV installation if the Ruby version matches.
#####

load("@bazel_skylib//:workspace.bzl", "bazel_skylib_workspace")
bazel_skylib_workspace()

rules_ruby_select_sdk(version = "3.0.2")

#####
# Now, load the ruby_bundle rule & install gems specified in the Gemfile
#####

load(
    "@bazel_ruby_rules_ruby//ruby:defs.bzl",
    "ruby_bundle",
)

ruby_bundle(
```

```

Ê   name = "bundle",
Ê   # Specify additional paths to be loaded from the gems at runtime, if any.
Ê   # Since spec.require_paths in Gem specifications are auto-included, directory
paths
Ê   # in spec.require_paths do not need to be listed in includes hash.
Ê   includes = {
Ê       "grpc": ["etc"],
Ê   },
Ê   excludes = {
Ê       "mini_portile": ["test/**/*"],
Ê   },
Ê   gemfile = "://:Gemfile",
Ê   gemfile_lock = "://:Gemfile.lock",
Ê   )

# You can specify more than one bundle in the WORKSPACE file
ruby_bundle(
Ê   name = "bundle_app_shopping",
Ê   gemfile = "://:apps/shopping/Gemfile",
Ê   gemfile_lock = "://:apps/shopping/Gemfile.lock",
Ê   )

# You can also install from Gemfile using `gemspec`.
ruby_bundle(
Ê   name = "bundle_gemspec",
Ê   srcs = ["://:lib/my_gem/my_gem.gemspec"],
Ê   gemfile = "://:lib/my_gem/Gemfile",
Ê   gemfile_lock = "://:lib/my_gem/Gemfile.lock",
Ê   )

```

4.2. BUILD.bazel file(s)

Any of the project BUILD files can now reference any gems included in the Gemfile referenced by the ruby_bundle rule, and defined in the project's WORKSPACE file.

4.2.1. Define Ruby Executable, Library and an RSpec

Add ruby_library, ruby_binary, ruby_rspec or ruby_test into your BUILD.bazel files.


```

# Define Ruby executable, test, spec and package a gem
#

load(
  "@bazel ruby_rules_ruby//ruby:defs.bzl",
  "ruby_binary",
  "ruby_library",
  "ruby_test",
  "ruby_rspec",
)

ruby_library(
  name = "foo",
  srcs = glob(["lib/**/*.rb"]),
  includes = ["lib"],
  deps = [
    "@bundle//:activesupport",
    "@bundle//:awesome_print",
    "@bundle//:rubocop",
  ]
)

ruby_binary(
  name = "bar",
  srcs = ["bin/bar"],
  deps = [":foo"],
)

ruby_test(
  name = "foo-test",
  srcs = ["test/foo_test.rb"],
  deps = [":foo"],
)

ruby_rspec(
  name = "foo-spec",
  specs = glob(["spec/**/*.rb"]),
  rspec_args = { "--format": "progress" },
  deps = [":foo"]
}

```

4.2.2. Package Ruby files as a Gem

Use `ruby_gem` rule to package any number of ruby files or folders into a Ruby-Gem compatible ZIP archive.

```
load(
  "@bazel_ruby_rules_ruby//ruby:defs.bzl",
  "ruby_gem",
)

ruby_gem(
  name = "awesome-sauce-gem", # name of the build target
  gem_name = "awesome-sauce", # name of the gem
  gem_version = "0.1.0",
  gem_summary = "Example gem to demonstrate Bazel Gem packaging",
  gem_description = "Example gem to demonstrate Bazel Gem packaging",
  gem_homepage = "https://github.com/bazelruby/rules_ruby",
  gem_authors = [
    "Bazel Ruby",
    "Konstantin Gredeskoul",
  ],
  gem_author_emails = [
    "bazelruby@googlegroups.com",
  ],
  gem_runtime_dependencies = {
    "colored2": "~> 3.1.2",
    "hashie": "",
  },
  gem_development_dependencies = {
    "rspec": "",
    "rspec-its": "",
    "rubocop": "",
  },
  srcs = [
    glob("{bin,exe,lib,spec}/**/*.rb"),
  ],
  deps = [
    "//lib:example_gem",
  ],
)
```

4.3. Tool Specific Setup

4.3.1. ASDF

If you are using ASDF to manage your ruby installs, you can use them by adding `.bazelrc`:

```
build --test_env=ASDF_DIR --test_env=ASDF_DATA_DIR
build --action_env=ASDF_DIR --test_env=ASDF_DATA_DIR
```

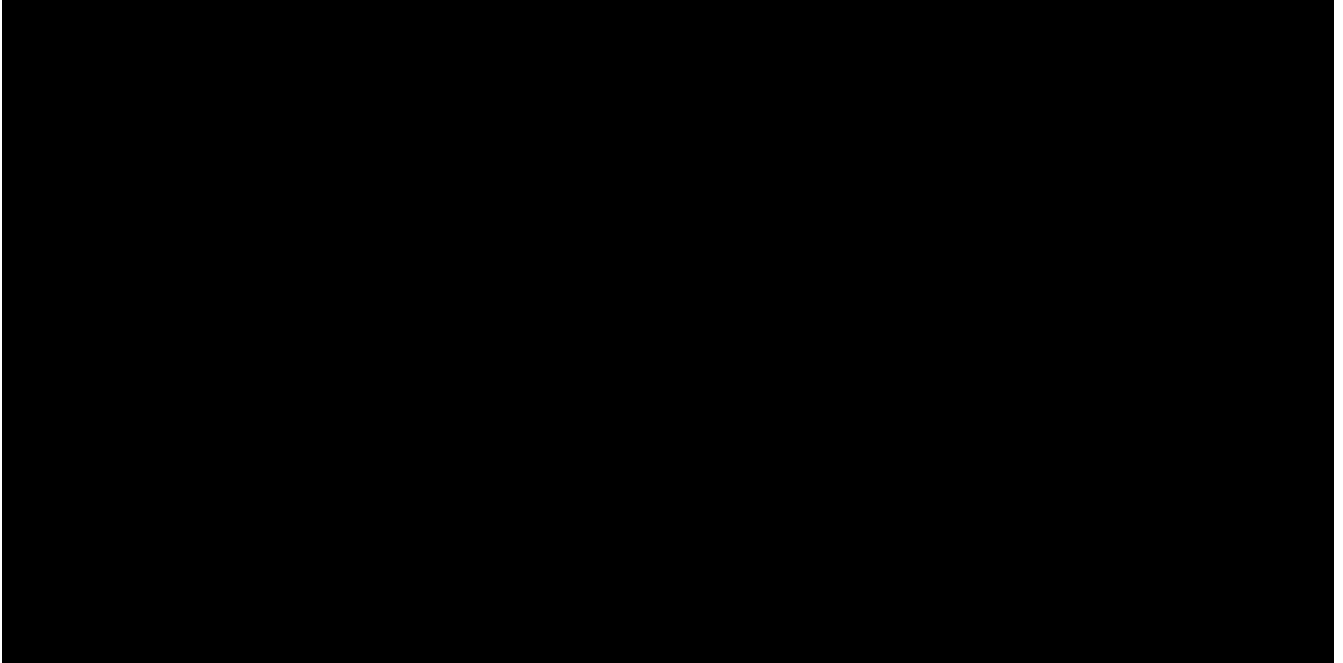
You will have to be sure to export the `ASDF_DATA_DIR` in your profile since it's not set by default. e.g.
`export ASDF_DATA_DIR="$HOME/.asdf"`

4.4. Rule Dependency Diagram



this diagram is somewhat outdated.

The following diagram attempts to capture the implementation behind `ruby_library` that depends on the result of `bundle install`, and a `ruby_binary` that depends on both:



Chapter 5. Rules

5.1. ruby_library

```
ruby_library(  
  Ê name,  
  Ê deps,  
  Ê srcs,  
  Ê data,  
  Ê compatible_with,  
  Ê deprecation,  
  Ê distribs,  
  Ê features,  
  Ê licenses,  
  Ê restricted_to,  
  Ê tags,  
  Ê testonly,  
  Ê toolchains,  
  Ê visibility)
```

Attributes	
name	<p>Name, required</p> <p>A unique name for this rule.</p>
srcs	<p>List of Labels, optional</p> <p>List of <code>.rb</code> files.</p> <p>At least <code>srcs</code> or <code>deps</code> must be present</p>
deps	<p>List of Labels, optional</p> <p>List of targets that are required by the <code>srcs</code> Ruby files.</p> <p>At least <code>srcs</code> or <code>deps</code> must be present</p>
includes	<p>List of strings, optional</p> <p>List of paths to be added to <code>\$LOAD_PATH</code> at runtime. The paths must be relative to the the workspace which this rule belongs to.</p>
rubyopt	<p>List of strings, optional</p> <p>List of options to be passed to the Ruby interpreter at runtime.</p> <div><div>!</div><div>-I option should usually go to <code>includes</code> attribute.</div></div>

Attributes	
And other <i>common attributes</i> .	

5.2. ruby_binary

```

ruby_binary(
  name,
  deps,
  srcs,
  data,
  main,
  compatible_with,
  deprecation,
  distribs,
  features,
  licenses,
  restricted_to,
  tags,
  testonly,
  toolchains,
  visibility,
  args,
  output_licenses
)

```

Attributes	
name	<p>Name, required</p> <p>A unique name for this rule.</p>
srcs	<p>List of Labels, required</p> <p>List of .rb files.</p>
deps	<p>List of Labels, optional</p> <p>List of targets that are required by the srcs Ruby files.</p>
main	<p>Label, optional</p> <p>The entrypoint file. It must be also in srcs.</p> <p>If not specified, \$(NAME).rb where \$(NAME) is the name of this rule.</p>

Attributes	
<code>includes</code>	<p>List of strings, optional</p> <p>List of paths to be added to <code>\$LOAD_PATH</code> at runtime. The paths must be relative to the workspace which this rule belongs to.</p>
<code>rubyopt</code>	<p>List of strings, optional</p> <p>List of options to be passed to the Ruby interpreter at runtime.</p> <p>! -I option should usually go to <code>includes</code> attribute.</p>
And other <i>common attributes</i> .	

5.3. `ruby_test`

```

ruby_test(
  Ê  name,
  Ê  deps,
  Ê  srcs,
  Ê  data,
  Ê  main,
  Ê  compatible_with,
  Ê  deprecation,
  Ê  distribs,
  Ê  features,
  Ê  licenses,
  Ê  restricted_to,
  Ê  tags,
  Ê  testonly,
  Ê  toolchains,
  Ê  visibility,
  Ê  args,
  Ê  size,
  Ê  timeout,
  Ê  flaky,
  Ê  local,
  Ê  shard_count
)

```

Attributes	
<code>name</code>	<p>Name, required</p> <p>A unique name for this rule.</p>

Attributes	
<code>srcs</code>	<p>List of Labels, required</p> <p>List of <code>.rb</code> files.</p>
<code>deps</code>	<p>List of Labels, optional</p> <p>List of targets that are required by the <code>srcs</code> Ruby files.</p>
<code>main</code>	<p>Label, optional</p> <p>The entrypoint file. It must be also in <code>srcs</code>.</p> <p>If not specified, <code>\$(NAME).rb</code> where <code>\$(NAME)</code> is the <code>name</code> of this rule.</p>
<code>includes</code>	<p>List of strings, optional</p> <p>List of paths to be added to <code>\$LOAD_PATH</code> at runtime. The paths must be relative to the the workspace which this rule belongs to.</p>
<code>rubyopt</code>	<p>List of strings, optional</p> <p>List of options to be passed to the Ruby interpreter at runtime.</p> <p>! <code>-I</code> option should usually go to <code>includes</code> attribute.</p>
And other common attributes .	

5.4. `ruby_bundle`

NOTE: This is a repository rule, and can only be used in a `WORKSPACE` file.

This rule installs gems defined in a Gemfile using Bundler, and exports individual gems from the bundle, as well as the entire bundle, available as a `ruby_library` that can be depended upon from other targets.

```
ruby_bundle(
  name,
  gemfile,
  gemfile_lock,
  bundler_version = "2.1.4",
  includes = {},
  excludes = {},
  srcs = [],
  vendor_cache = False,
  ruby_sdk = "@org_ruby_lang_ruby_toolchain",
  ruby_interpreter = "@org_ruby_lang_ruby_toolchain//:ruby",
)
```

Attributes	
name	<p>Name, required</p> <p>A unique name for this rule.</p>
gemfile	<p>Label, required</p> <p>The Gemfile which Bundler runs with.</p>
gemfile_lock	<p>Label, optional</p> <p>The Gemfile.lock which Bundler runs with.</p> <div data-bbox="389 629 405 685">!</div> <p>This rule never updates the Gemfile.lock. It is your responsibility to generate/update Gemfile.lock</p>
srcs	<p>List of Labels, optional</p> <p>List of additional files required for Bundler to install gems. This could usually include *.gemspec files.</p>
vendor_cache	<p>Bool, optional</p> <p>Symlink the vendor directory into the Bazel build space, this allows Bundler to access vendored Gems</p>
bundler_version	<p>String, optional</p> <p>The Version of Bundler to use. Defaults to 2.1.4.</p> <div data-bbox="389 1312 405 1368">!</div> <p>This rule never updates the Gemfile.lock. It is your responsibility to generate/update Gemfile.lock</p>
includes	<p>Dictionary of key-value-pairs (key: string, value: list of strings), optional</p> <p>List of glob patterns per gem to be additionally loaded from the library. Keys are the names of the gems which require some file/directory paths not listed in the require_paths attribute of the gems specs to be also added to \$LOAD_PATH at runtime. Values are lists of blob path patterns, which are relative to the root directories of the gems.</p>
excludes	<p>Dictionary of key-value-pairs (key: string, value: list of strings), optional</p> <p>List of glob patterns per gem to be excluded from the library. Keys are the names of the gems. Values are lists of blob path patterns, which are relative to the root directories of the gems. The default value is ["/*.", "/* /"]</p>
And other common attributes.	

5.4.1. Conventions

`ruby_bundle` creates several targets that can be used downstream. In the examples below we assume that your `ruby_bundle` has a name `app_bundle`:

- ¥ `@app_bundle//:bundler` references just the Bundler from the bundle.
- ¥ `@app_bundle//:gems` references *all* gems in the bundle (i.e. "the entire bundle").
- ¥ `@app_bundle//:gem-name` references *just the specified* gem in the bundle, eg. `@app_bundle//:awesome_print`.
- ¥ `@app_bundle//:bin` references to all installed executables from this bundle, with individual executables accessible via eg. `@app_bundle//:bin/rubocop`

5.4.2. WORKSPACE:

```
load("@bazel_ruby_rules_ruby//ruby:defs.bzl", "ruby_bundle")

ruby_bundle(
    name = "gems",
    bundler_version = '2.1.4',
    gemfile = "//:Gemfile",
    gemfile_lock = "//:Gemfile.lock",
)
```

Vendor directory handling

To use the vendor cache, you have to declare a `managed_directory` in your workspace. The name should match the name of the bundle.

```
load("@bazel_ruby_rules_ruby//ruby:defs.bzl", "ruby_bundle")

workspace(
    name = "my_wksp",
    managed_directories = {"@bundle": ["vendor"]},
)

ruby_bundle(
    name = "bundle",
    bundler_version = "2.1.2",
    vendor_cache = True,
    gemfile = "//:Gemfile",
    gemfile_lock = "//:Gemfile.lock",
)
```

5.4.3. BUILD.bazel:

```
# Reference the entire bundle with :gems

ruby_library(
  name = "foo",
  srcs = ["foo.rb"],
  deps =["@gems//:gems"],
)

# Or, reference specific gems from the bundle like so:

ruby_binary(
  name = "rubocop",
  srcs = [":foo", ".rubocop.yml"],
  args = ["-P", "-D", "-c" ".rubocop.yml"],
  main = "@gems//:bin/rubocop",
  deps =["@gems//:rubocop"],
)
```

5.5. ruby_rspec

```
ruby_rspec(
  name,
  deps,
  srcs,
  data,
  main,
  rspec_args,
  bundle,
  compatible_with,
  deprecation,
  distribs,
  features,
  licenses,
  restricted_to,
  tags,
  testonly,
  toolchains,
  visibility,
  args,
  size,
  timeout,
  flaky,
  local,
  shard_count
)
```

Attributes	
name	<p>Name, required</p> <p>A unique name for this rule.</p>
srcs	<p>List of Labels, required</p> <p>List of <code>.rb</code> files.</p>
deps	<p>List of Labels, optional</p> <p>List of targets that are required by the <code>srcs</code> Ruby files.</p>
main	<p>Label, optional</p> <p>The entrypoint file. It must be also in <code>srcs</code>.</p> <p>If not specified, <code>\$(NAME).rb</code> where <code>\$(NAME)</code> is the <code>name</code> of this rule.</p>
rspec_args	<p>List of strings, optional</p> <p>Command line arguments to the <code>rspec</code> binary, eg <code>["--progress", "-p2", "-b"]</code></p> <p>If not specified, the default arguments defined in <code>constants.bzl</code> are used: <code>--format=documentation --force-color</code>.</p>
includes	<p>List of strings, optional</p> <p>List of paths to be added to <code>\$LOAD_PATH</code> at runtime. The paths must be relative to the the workspace which this rule belongs to.</p>
rubyopt	<p>List of strings, optional</p> <p>List of options to be passed to the Ruby interpreter at runtime.</p> <div> <p>!</p> <p>-I option should usually go to <code>includes</code> attribute.</p> </div>
And other common attributes .	

5.6. `ruby_gem`

Used to generate a zipped gem containing its `srcs`, dependencies and a `gemspec`.

```

ruby_gem(
  name,
  gem_name,
  gem_version,
  gem_summary,
  gem_description,
  gem_homepage,
  gem_authors,
  gem_author_emails,
  gem_runtime_dependencies,
  gem_development_dependencies,
  require_paths = ["lib"],
  srcs = srcs,
  deps = deps,
  data = data
)

```

Attributes	
name	<p>Name, required</p> <p>A unique name for this build target.</p>
gem_name	<p>Name of the gem, required</p> <p>The name of the gem to be generated.</p>
gem_version	<p>String, optional</p> <p>The version of the gem. Is used to name the output file, which becomes <code>name-version.zip</code>, and also included in the Gemspec.</p>
gem_summary	<p>String, optional</p> <p>One line summary of the gem purpose.</p>
gem_description	<p>String, required</p> <p>Single-line, paragraph-sized description text for the gem.</p>
gem_homepage	<p>String, optional</p> <p>Homepage URL of the gem.</p>
gem_authors	<p>List of Strings, required</p> <p>List of human readable names of the gem authors. Required to generate a valid gemspec.</p>

Attributes	
<code>gem_author_emails</code>	<p>List of Strings, optional</p> <p>List of email addresses of the authors.</p>
<code>srcs</code>	<p>List of Labels, optional</p> <p>List of <code>.rb</code> files.</p> <p>At least <code>srcs</code> or <code>deps</code> must be present</p>
<code>deps</code>	<p>List of Labels, optional</p> <p>List of targets that are required by the <code>srcs</code> Ruby files.</p> <p>At least <code>srcs</code> or <code>deps</code> must be present</p>
<code>require_paths</code>	<p>List of Strings, optional</p> <p>List of paths to be added to the Ruby <code>LOAD_PATH</code> when using this gem. Typically this value is just <code>lib</code> (which is also the default).</p>
<code>gem_runtime_dependencies</code>	<p>String Dictionary, optional</p> <p>This is a dictionary where keys are gem names, and values are either an empty string or a gem version specification. For instance, the pessimistic version specifier <code>~> 3.0</code> means that all versions up to <code>4.0</code> are accepted.</p>
<code>gem_development_dependencies</code>	<p>String Dictionary, optional</p> <p>Similar to the above, this specifies gems necessary for the development of the above gem, such as testing gems, linters, code coverage and more.</p>
And other common attributes .	

Chapter 6. Potential Future Features

" Using various versions of Ruby installed locally

Building native extensions in gems with Bazel

Releasing your gems with Bazel ([Coinbase fork](#) might have this feature, worth checking)

Chapter 7. Contributing

We welcome contributions to RulesRuby. Please make yourself familiar with the [code of conduct](#), which basically says “don’t be an a-hole”.

You may notice that there is more than one Bazel WORKSPACE inside this repo. There is one in [examples/simple_script](#) for instance, because we use this example to validate and test the rules. So be mindful whether your current directory contains [WORKSPACE](#) file or not.

7.1. Setup

7.1.1. Using the Script

You will need Homebrew installed prior to running the script.

After that, cd into the top level folder and run the setup script in your Terminal:

```
! bin/setup
```

This runs a complete setup, shouldn’t take too long. You can explore various script options with the [help](#) command:

```
! bin/setup -h
```

USAGE

```
Ê # without any arguments runs a complete setup.
```

```
Ê bin/setup
```

```
Ê # alternatively, a sub-setup function name can be passed:
```

```
Ê bin/setup [ gems | git-hook | help | main | os-specific | rbenv | remove-git-hook ]
```

DESCRIPTION:

```
Ê Runs full setup without any arguments.
```

```
Ê Accepts one optional argument – one of the actions that typically run  
Ê as part of setup, with one exception – remove-git-hook.
```

```
Ê This action removes the git commit hook installed by the setup.
```

EXAMPLES:

```
Ê bin/setup
```

```
Ê Or, to run only one of the sub-functions (actions), pass  
Ê it as an argument:
```

```
Ê bin/setup help
```

```
Ê bin/setup remove-git-hook
```

7.1.2. OS-Specific Setup

Note that the setup contains `os-specific` section. This is because there are two extension scripts:

```
¥ bin/setup-linux
```

```
¥ bin/setup-darwin
```

Those will install Bazel and everything else you need on either platform. In fact, we use the linux version on CI.

7.2. Verifying Your Environment

We provided a handy script `bin/show-env` to display where your dependencies are coming from. Here is an example of running it on a Mac OS-X system:

```
! bin/show-env
```

7.2.1. Issues During Setup

Please report any errors to `bin/setup` as Issues on Github. You can assign them to @kigster. If I am not responding fast enough, and you are in a hurry, please email kigster AT gmail directly.

7.3. Developing Rules

Besides making yourself familiar with the existing code, and [Bazel documentation on writing rules](#), you might want to follow this order:

1. Setup dev tools as described in the [setup](#) section.
2. hack, hack, hack!
3. Make sure all tests pass. You can run a single command for that (but see more on it [below](#)).

```
bin/test-suite
```


OR, you can run individual Bazel test commands from the inside.

```
¥ bazel test //
```

```
¥ cd examples/simple_script && bazel test //
```

1. Open a pull request in Github, and please be as verbose as possible in your description.

In general, it's always a good idea to ask questions first. You can do so by creating an issue.

7.4. Running Tests

After running setup, and since this is a bazel repo you can use Bazel commands:

```
bazel build //...:all
bazel query //...:all
bazel test //...:all
```

But to run tests inside each sub-WORKSPACE, you will need to repeat that in each sub-folder. Luckily, there is a better way.

7.4.1. Test Script

This script runs all tests (including sub-workspaces) when ran without arguments:

```
bin/test-suite
```

Run it with `help` command to see other options, and to see what parts you can run individually. At the moment they are:

```
# alternatively, a partial test name can be passed:
bin/test-suite [ all | bazel-info | buildifier | help | rspec | rubocop | simple-
script | workspace ]
```

On a MacBook Pro it takes about 3 minutes to run.

7.5. Linter

We are using RuboCop for ruby and Buildifier for Bazel. Both are represented by a single script `bin/linter`, which just like the scripts above runs ALL linters when ran without arguments, accepts `help` command, and can be run on a subset of linting strategies:

```
bin/linter
```

The following are the partial linting functions you can run:

```
# alternatively, a partial linter name can be passed:  
bin/linter [ all | buildifier | help | rubocop ]
```

7.6. Regenerating README.pdf & Changelog

To regenerate, first you may need to grab an [API token](#) and export the `GITHUB_TOKEN` variable:

```
export GITHUB_TOKEN=...
```

Then use the `make` target:

```
make update
```

Or, manually:

```
gem install github_changelog_generator  
github_changelog_generator -u bazel ruby -p rules_ruby -t your-github-token
```

Chapter 8. Copyright

© 2018-2021 BazelRuby Contributors.

Core Team:

¥ [Yuki Yugui Sonoda](#)

¥ [Konstantin Gredeskoul](#)

Core Team (Emeritus):

¥ [Graham Jenson](#)

Licensed under the [Apache License, Version 2.0](#) (the "License").

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.