
Scuba

Jonathon Reinhart

Mar 25, 2024

CONTENTS

1	Introduction	3
2	Installation	5
3	Configuration	7
4	Command-Line Interface	17
5	Bash Completion	19
6	Environment	21
7	Change Log	23
8	Contributing Guide	33

Local builds made easy, using Docker.

INTRODUCTION

Scuba makes it easier to use Docker containers in everyday development. It is intended to allow a developer to commit an environment setup where the entire build environment is encapsulated in a Docker container.

Its purpose is to lower the barrier to using Docker for everyday builds. Scuba keeps you from having to remember a complex `docker run` command line, and turns this:

```
$ docker run -it --rm -v $(pwd):/build:z -w /build -u $(id -u):$(id -g) gcc:5.1 make ↵  
↵myprogram
```

into this:

```
$ scuba make myprogram
```

Scuba references a `.scuba.yml` file which is intended to be checked-in to your project's version control, which ensures that all developers are always using the exact correct version of the the Docker build environment for a given commit.

INSTALLATION

2.1 Install via pip

Scuba is hosted at [PyPI](#), and installation via pip is the preferred method.

To install:

```
$ sudo pip install scuba
```

To install with `argcomplete` (for *Bash Completion* support):

```
$ sudo pip install scuba[argcomplete]
```

To uninstall:

```
$ sudo pip uninstall scuba
```

2.2 Install from source

Scuba can be built from source on Linux only (due to the fact that `scubainit` must be compiled):

1. Run `make` to build `scubainit`
2. Run `./run_unit_tests.py` to run the unit tests
3. Run `sudo python setup.py install` to install scuba
4. Run `./run_full_tests.py` to test the installed version of scuba

If `musl-libc` is installed, it can be used to reduce the size of `scubainit`, by overriding the `CC` environment variable in step 1:

```
CC=/usr/local/musl/bin/musl-gcc make
```

Note: Note that installing from source in this manner can lead to an installation with increased startup times for Scuba. See [#71](#) for more details. This can be remedied by forcing a `wheel` to be installed, as such:

```
export CC=/usr/local/musl/bin/musl-gcc    # (optional)
sudo pip install wheel
python setup.py bdist_wheel
sudo pip install dist/scuba-<version>-py3-none-any.whl
```

CONFIGURATION

Configuration is done using a [YAML](#) file named `.scuba.yml` in the root directory of your project. It is expected that `.scuba.yml` will be checked-in to version control.

An example `.scuba.yml` file might look like this:

```
image: gcc:5.1

aliases:
  build: make -j4
```

In this example, running `scuba build foo` would execute `make -j4 foo` in a `gcc:5.1` Docker container.

3.1 Scuba YAML File Reference

`.scuba.yml` is a [YAML](#) file which defines project-specific settings, allowing a project to use Scuba as part of manual command-line interaction. As with many other YAML file schemas, most options are controlled by top-level keys.

3.1.1 Top-level keys

Key	Scuba Version	Description	Alias
<i>image</i>	(all)	Docker image to run	Can override
<i>environment</i>	2.3.0	Environment variables	Can extend or override
<i>docker_args</i>	2.8.0	Additional arguments to <code>docker run</code>	Can extend or override
<i>volumes</i>	2.9.0	Additional volumes to mount	Can extend or override
<i>aliases</i>	1.1.0	Command/script aliases	
<i>hooks</i>	1.7.0	Hook scripts run during startup	
<i>shell</i>	2.6.0	Override container shell path	Can override
<i>entrypoint</i>	2.4.0	Override container ENTRYPOINT path	Can override

image

The `image` node defines the Docker image from which Scuba containers are created.

Example:

```
image: debian:8.2
```

The `image` node is usually necessary but, as of scuba 2.5, can be omitted for `.scuba.yml` files in which only the aliases are intended to be used.

environment

The optional `environment` node (*added in v2.3.0*) allows environment variables to be specified. This can be either a mapping (dictionary), or a list of `KEY=VALUE` pairs. If a value is not specified, the value is taken from the external environment.

Examples:

```
environment:
  FOO: "This is foo"
  SECRET:
```

```
environment:
- FOO=This is foo
- SECRET
```

docker_args

The optional `docker_args` node (*added in v2.8.0*) allows additional docker arguments to be specified.

Example:

```
docker_args: --privileged -v "/tmp/hello world:/tmp/hello world"
```

The value of `docker_args` is parsed as shell command line arguments using `shlex.split`.

The previous example could be equivalently written in YAML's *single-quoted style*:

```
docker_args: '--privileged -v "/tmp/hello world:/tmp/hello world"'
```

volumes

The optional `volumes` node (*added in v2.9.0*) allows additional *bind-mounts* to be specified. As of v2.13.0, *named volumes* are also supported.

`volumes` is a mapping (dictionary) where each key is the container-path. In the simple form, the value is a string, which can be:

- An absolute or relative path which results in a bind-mount.
- A Docker volume name.

Listing 1: Example of simple-form volumes

```
volumes:
  /var/lib/foo: /host/foo # bind-mount: absolute path
  /var/lib/bar: ./bar     # bind-mount: path relative to .scuba.yml dir
  /var/log: persist-logs  # named volume
```

In the complex form, the value is a mapping with the following supported keys:

- **hostpath**: An absolute or relative path specifying a host bind-mount.
- **name**: The name of a named Docker volume.
- **options**: A comma-separated list of volume options.

hostpath and name are mutually-exclusive and one must be specified.

Listing 2: Example of complex-form volumes

```
volumes:
  /var/lib/foo:
    hostpath: /host/foo # bind-mount
    options: ro,cached
  /var/log:
    name: persist-logs # named volume
```

The paths (host or container) used in volume mappings can contain environment variables **which are expanded in the host environment**. For example, this configuration would map the user's `/home/username/.config/application1` directory into the container at the same path.

```
volumes:
  $TEST_HOME/.config/application1: $TEST_HOME/.config/application1
```

If a referenced environment variable is not set, Scuba exits with a configuration error.

Volume container paths must be absolute.

Bind-mount host paths can be absolute or relative. If a relative path is used, it is interpreted as relative to the directory in which `.scuba.yml` is found. To avoid ambiguity with a named volume, relative paths must start with `./` or `../`.

Bind-mount host directories which do not already exist are created as the current user before creating the container.

Note: Because variable expansion is now applied to all volume paths, if one desires to use a literal `$` character in a path, it must be written as `$$`.

Note: Docker named volumes are created with `drwxr-xr-x` (0755) permissions. If scuba is not run with `--root`, the scuba user will be unable to write to this directory. As a workaround, one can use a [root hook](#) to change permissions on the directory.

```
volumes:
  /foo: foo-volume
hooks:
  root: chmod 777 /foo
```

aliases

The optional `aliases` node is a mapping (dictionary) of bash-like aliases, where each key is an alias, and each value is the command that will be run when that alias is specified as the *user command* during scuba invocation. The command is parsed like a shell command-line, and additional user arguments from the command line are appended to the alias arguments. Aliases follow the *common script schema*.

Example:

```
aliases:
  build: make -j4
```

In this example, `$ scuba build foo` would execute `make -j4 foo` in the container.

Aliases can also extend/override various top-level keys. See *Alias-level keys*.

hooks

The optional `hooks` node is a mapping (dictionary) of “hook” scripts that run as part of `scubainit` before running the user command. They use the *common script schema*. The following hooks exist:

- `root` - Runs just before `scubainit` switches from `root` to `scubauser`
- `user` - Runs just before `scubainit` executes the user command

Example:

```
hooks:
  root:
    script:
      - 'echo "HOOK: This runs before we switch users"'
      - id
  user: 'echo "HOOK: After switching users, uid=$(id -u) gid=$(id -g)''
```

shell

The optional `shell` node (*added in v2.6.0*) allows the default shell that Scuba uses in the container (`/bin/sh`) to be overridden by another shell. This is useful for images that do not have a shell located at `/bin/sh`.

Example:

```
shell: /busybox/sh
```

entrypoint

The optional `entrypoint` node (*added in v2.4.0*) allows the `ENTRYPOINT` of the Docker image to be overridden:

```
entrypoint: /another/script
```

The entrypoint can also be set to null, which is useful when an image’s entrypoint is not suitable:

```
entrypoint:
```

3.1.2 Alias-level keys

Key	Scuba Version	Description
<i>image</i>	1.1.0	Override Docker image to run
<i>environment</i>	2.3.0	Extend / override environment variables
<i>docker_args</i>	2.8.0	Extend / override additional arguments to <code>docker run</code>
<i>volumes</i>	2.9.0	Extend / override additional volumes to mount
<i>shell</i>	2.6.0	Override container shell path
<i>entrypoint</i>	2.4.0	Override container ENTRYPOINT path
<i>root</i>	2.6.0	Run container as root

image

Aliases can override the global image, allowing aliases to use different images. Example:

```
image: default_image
aliases:

# This one inherits the default, top-level 'image' and specifies "script" as a string
default:
  script: cat /etc/os-release

# This one specifies a different image to use and specifies "script" as a list
different:
  image: alpine
  script:
    - cat /etc/os-release
```

environment

Aliases can add to the top-level environment and override its values using the same syntax:

```
environment:
  FOO: "Top-level"
aliases:
  example:
    environment:
      FOO: "Override"
      BAR: "New"
    script:
      - echo $FOO $BAR
```

docker_args

Aliases can extend the top-level `docker_args`. The following example will produce the docker arguments `--privileged -v /tmp/bar:/tmp/bar` when executing the `example` alias:

```
docker_args: --privileged
aliases:
  example:
    docker_args: -v /tmp/bar:/tmp/bar
    script:
      - ls -l /tmp/
```

Aliases can also opt to override the top-level `docker_args`, replacing it with a new value. This is achieved with the `!override` tag:

```
docker_args: -v /tmp/foo:/tmp/foo
aliases:
  example:
    docker_args: !override -v /tmp/bar:/tmp/bar
    script:
      - ls -l /tmp/
```

The content of the `docker_args` key is re-parsed as YAML in order to allow combining the `!override` tag with other tags; however, this requires quoting the value, since YAML forbids a plain-style scalar from beginning with a `!` (see [the spec](#)). In the next example, the top-level alias is replaced with an explicit `!!null` tag, so that no additional arguments are passed to docker when executing the `example` alias:

```
docker_args: -v /tmp/foo:/tmp/foo
aliases:
  example:
    docker_args: !override '!!null'
    script:
      - ls -l /tmp/
```

volumes

Aliases can extend or override the top-level volumes:

```
volumes:
  /var/lib/foo: /host/foo
aliases:
  example:
    volumes:
      /var/lib/foo: /example/foo
      /var/lib/bar: /example/bar
    script:
      - ls -l /var/lib/foo /var/lib/bar
```

shell

Aliases can override the shell from the default or the top-level of the `.scuba.yml` file:

```
aliases:
  my_shell:
    shell: /bin/cool_shell
    script:
      - echo "This is executing in cool_shell"
  busybox_shell:
    script:
      - echo "This is executing in scuba's default shell"
```

entrypoint

An alias can override the image-default or top-level `.scuba.yml` entrypoint, which is most useful when an alias defines a special image.

```
aliases:
  build:
    image: build/image:1.2
    entrypoint:
```

root

The optional `root` node (*added in v2.6.0*) allows an alias to specify whether its container should be run as root:

```
aliases:
  root_check:
    root: true
    script:
      - echo 'Only root can do this!'
      - echo "I am UID $(id -u)"
      - cat /etc/shadow
```

3.1.3 Common script schema

Several parts of `.scuba.yml` which define “scripts” use a common schema. The *common script schema* can define a “script” in one of several forms:

The *simple* form is simply a single string value:

```
hooks:
  user: echo hello
```

The *complex* form is a mapping, which must contain a `script` subkey, whose value is either single string value:

```
hooks:
  root:
    script: echo hello
```

... or a list of strings making up the script:

```
hooks:
  root:
    script:
      - 'echo hello!'
      - touch foo
      - 'echo goodbye :-('
```

Note that in any case, YAML strings do not need to be enclosed in quotes, unless there are “confusing” characters (like a colon). In any case, it is always safer to include quotes.

3.1.4 Accessing external YAML content

In addition to normal [YAML](#) syntax, an additional constructor, `!from_yaml`, (*added in v1.2.0*) is available for use in `.scuba.yml` which allows a value to be retrieved from an external YAML file. It has the following syntax:

```
!from_yaml filename key
```

Arguments:

- `filename` - The path of an external YAML file (relative to `.scuba.yml`)
- `key` - A dot-separated locator of the key to retrieve

This is useful for projects where a Docker image in which to build is already specified in another YAML file, for example in `.gitlab-ci.yml`. This eliminates the redundancy between the configuration files. An example which uses this:

Listing 3: `.gitlab-ci.yml`

```
image: gcc:5.1
```

Listing 4: `.scuba.yml`

```
image: !from_yaml .gitlab-ci.yml image
```

Here’s a more elaborate example which defines multiple aliases which correspond to jobs defined by `.gitlab-ci.yml`:

Listing 5: `.gitlab-ci.yml`

```
build_c:
  image: gcc:5.1
  script:
    - make something
    - make something-else

build_py:
  image: python:3.7
  script:
    - setup.py bdist_wheel
```

Listing 6: `.scuba.yml`

```
# Note that 'image' is not necessary if only invoking aliases
```

(continues on next page)

(continued from previous page)

```
aliases:
  build_c:
    image: !from_yaml .gitlab-ci.yml build_c.image
    script: !from_yaml .gitlab-ci.yml build_c.script
  build_py:
    image: !from_yaml .gitlab-ci.yml build_py.image
    script: !from_yaml .gitlab-ci.yml build_py.script
```

An easier but less-flexible method is to simply import the entire job's definition. This works because Scuba ignores unrecognized keys in an alias:

Listing 7: .scuba.yml

```
aliases:
  build_c: !from_yaml .gitlab-ci.yml build_c
  build_py: !from_yaml .gitlab-ci.yml build_py
```

This example which concatenates two jobs from .gitlab-ci.yml into a single alias. This works by flattening the effective script node that results by including two elements that are lists.

Listing 8: .gitlab-ci.yml

```
image: gcc:5.1

part1:
  script:
    - make something
part2:
  script:
    - make something-else
```

Listing 9: .scuba.yml

```
image: !from_yaml .gitlab-ci.yml image

aliases:
  all_parts:
    script:
      - !from_yaml .gitlab-ci.yml part1.script
      - !from_yaml .gitlab-ci.yml part2.script
```

Dots (.) in a YAML *path* can be escaped using a backslash (which must be doubled inside of quotes). This example shows how to reference job names containing a . character:

Listing 10: .gitlab-ci.yml

```
image: gcc:5.1

.part1:
  script:
    - make something
.part2:
  script:
```

(continues on next page)

(continued from previous page)

```
- make something-else
```

Listing 11: .scuba.yml

```
image: !from_yaml .gitlab-ci.yml image

aliases:
  build_part1: !from_yaml .gitlab-ci.yml "\\part1.script"
  build_part2: !from_yaml .gitlab-ci.yml "\\part2.script"
```

Additional examples can be found in the `example` directory.

COMMAND-LINE INTERFACE

```
scuba [-h]
      [-d DOCKER_ARG] [-e ENV_VAR] [--entrypoint ENTRYPOINT]
      [--image IMAGE] [--shell SHELL] [-n] [-r] [-v] [-V]
      COMMAND... | ALIAS...
```

Positional Arguments:

COMMAND

The command (and arguments) to run in the container

ALIAS

Alternatively, an *alias* to run

Options:

- h, --help** Show help message and exit
- d DOCKER_ARG, --docker-arg DOCKER_ARG** Pass additional arguments to docker run. These are appended to any *docker_args* from *.scuba.yml*.
DOCKER_ARG is the full argument to `docker run`. *Note:* The `-` in the *DOCKER_ARG* can confuse scuba's argument parsing. The solution is to use an equal sign: `-d='--cpus=2'`
This argument can be given multiple times.
- e ENV_VAR, --env ENV_VAR** Environment variables to pass to docker. These are merged with (and override) any *environment* variables from *.scuba.yml*.
ENV_VAR is given as `KEY=value`.
This argument can be given multiple times.
- entrypoint ENTRYPOINT** Override the default ENTRYPOINT of the image
- image IMAGE** Override Docker image specified in *.scuba.yml*
- shell SHELL** Override shell used in Docker container
- n, --dry-run** Don't actually invoke docker; just print the docker cmdline
- r, --root** Run container as root
- v, --version** Show scuba version and exit
- V, --verbose** Be verbose

BASH COMPLETION

Scuba supports command-line completion using the [argcomplete](#) package. Per the [argcomplete README](#), command-line completion can be activated by:

- Running `eval "$(register-python-argcomplete scuba)"` manually to enable completion *in the current shell instance*
- Adding `eval "$(register-python-argcomplete scuba)"` to `~/.bash_completion`
- Running `activate-global-python-argcomplete --user` to install the script `~/.bash_completion.d/python-argcomplete`.

Note: The generated file must be sourced, which is *not* the default behavior. Adding the following code block to `~/.bash_completion` is one possible solution:

```
for bcfile in ~/.bash_completion.d/*; do
    [ -f "$bcfile" ] && . "$bcfile"
done
```

-
- Running `activate-global-python-argcomplete` as root (or `sudo`) to use `argcomplete` for *all* users

ENVIRONMENT

Scuba defines the following environment variables in the container:

- `SCUBA_ROOT` – (*added in v2.4.0*) The root of the scuba working directory mount; the directory where `.scuba.yml` was found

CHANGE LOG

All notable changes to this project will be documented in this file. This project adheres to [Semantic Versioning](#).

7.1 2.13.0 - 2024-03-25

7.1.1 Added

- Added support for Python 3.12 (#244)
- Add explicit support for mounting named volumes (#250)
 - This officially restores and extends the (unsupported) pre-v2.12 behavior.

7.1.2 Changed

- Removed use of deprecated `pkg_resources` (#247)
- Rewrote `scubainit` in Rust (#232)

7.2 2.12.0 - 2023-09-15

7.2.1 Added

- Enable the use of relative paths in a volume `hostpath` (#227)

7.3 2.11.0 - 2023-09-09

7.3.1 Changed

- Introduced `pyproject.toml` and moved metadata from `setup.py` (#211)
- Added type annotations to `scuba` package and `mypy` checking in CI (#207)

7.3.2 Removed

- Drop support for Python 3.5 - 3.6 (#205)

7.3.3 Fixed

- Fixed bug causing invalid volume spec error on Docker 24.0.5 and newer (#217)

7.4 2.10.1 - 2023-03-07

7.4.1 Fixed

- Create directories for volumes as invoking user rather than root. (#201)

7.5 2.10.0 - 2022-01-12

7.5.1 Added

- Add ability to use environment variables in volume paths (#192)

7.6 2.9.0 - 2021-09-15

7.6.1 Added

- Add ability to specify volumes in `.scuba.yml` (#186)

7.7 2.8.0 - 2021-08-18

7.7.1 Added

- Add ability to specify additional docker arguments in `.scuba.yml` (#177)

7.7.2 Changed

- Switched testing framework from from nose to pytest

7.8 2.7.0 - 2020-06-08

7.8.1 Changed

- Switched to using `argcomplete` to provide Bash command line completion (#162)

7.9 2.6.1 - 2020-04-24

7.9.1 Fixed

- `scubainit` ignores matching `passwd/group/shadow` file entries when creating the user. This allows transparently running `scuba` as root. (#164)
- Fixed bug where `scubainit` incorrectly displayed the exit status of a failed hook script. (#165)
- Fixed bug where user home directory was not writable when `scuba workdir` existed below the home directory. (#169)

7.10 2.6.0 - 2020-03-25

7.10.1 Added

- Add ability to override the shell in which the `scuba`-generated script is run, via command line option (`--shell`) or via `.scuba.yml` (#159)
- Add ability to specify in `.scuba.yml` that a particular alias should run as root (#159)

7.11 2.5.0 - 2020-03-05

7.11.1 Changed

- Use `username/groupname` of invoking user inside container (#153)
- Ignore already existing `UID/GIDs` (#139)
- Allow top-level `image` to be omitted in `.scuba.yml` (#158)

7.11.2 Fixed

- Fixed deprecation error with `collections.Mapping` (#156)

7.11.3 Removed

- Drop support for Python 2 (#154)

7.12 2.4.2 - 2020-02-24

7.12.1 Changed

- Use GitHub Actions instead of Travis CI for publishing releases

7.13 2.4.1 - 2020-02-21

7.13.1 Added

- Cache yaml files loaded by `!from_yaml`

7.13.2 Removed

- Drop support for Python 3.4

7.14 2.4.0 - 2020-01-06

7.14.1 Added

- Enable scuba to override entrypoint via `--entrypoint` or `.scuba.yml` (#125)
- Add support for nested scripts (#128)
- Add `SCUBA_ROOT` environment variable (#129)
- Add support for escaped dots in `!from_yaml` (#137)

7.14.2 Changed

- Don't run image entrypoint for each line in a multi-line alias (#121)
- Use `yaml.SafeLoader` for loading config (#133)

7.14.3 Removed

- Drop support for Python 2.6, 3.2, and 3.3 (#119, #130)

7.15 2.3.0 - 2018-09-10

7.15.1 Added

- Add `-e/--env` command-line option (#111)
- Add support for setting environment in `.scuba.yml` (#120)

7.15.2 Changed

- Implemented auto-versioning using Git and Travis (#112)

7.15.3 Fixed

- Copy `scubainit` to allow SELinux relabeling (#117)

7.16 2.2.0 - 2018-03-07

7.16.1 Changed

- Allow `script` to be a single string value in the “common script schema” which applies to hooks and aliases (#102)

7.16.2 Fixed

- Display nicer error message if no command is given and image doesn't specify a `Cmd` (#104)
- Don't mangle `&&` in scripts (#100)
- Don't allocate `tty` if `stdin` is redirected (#95)

7.17 2.1.0 - 2017-04-03

7.17.1 Added

- Added `--image` option (#87)

7.18 2.0.1 - 2017-01-17

7.18.1 Fixed

- Fixed image entrypoint being ignored (#83)

7.19 2.0.0 - 2016-11-21

7.19.1 Added

- Added support for enhanced aliases (#67)
- Added support for per-alias image specification (#68)
- Add bash completion support (#69)

7.19.2 Changed

- All ancillary files are bind-mounted via single temp dir
- Hook scripts are moved to hooks/ subdirectory
- User commands always executed via shell (#66)
- Top-level directory mounted at same path in container (#70)
- Alias names cannot contain spaces
- Improve distributions (#74, #75, #76, #78)

7.19.3 Removed

- Remove support for remote Docker instances (#64) Support for this was limited/broken as of 1.7.0 anyway; this officially removes support for it.

7.19.4 Fixed

- Fixed inability to run an image that doesn't yet exist locally, broken in 1.7.0 (#79)

7.20 1.7.0 - 2016-05-19

7.20.1 Added

- Add support for scubainit hooks

7.20.2 Changed

- `scubainit` re-implemented as a C program, which does the following:
 - Creates the `scubauser` user/group
 - Sets the `umask`
 - Switches users then *execs* the user command This is to provide more control during initialization, without the artifacts caused by the use of ‘su’ in the `.scubainit` from 1.3.
- `scubauser` now has a proper writable home directory in the container (#45)

7.21 1.6.0 - 2016-02-06

7.21.1 Added

- Add `-d` to pass arbitrary arguments to `docker run`

7.22 1.5.0 - 2016-02-01

7.22.1 Added

- Add `-r` option to run container as root
- Add automated testing (both unit and system tests)
- Add support for Python 2.6 - 3.5
- Added to PyPI

7.22.2 Changed

- Scuba is now a package, and `setup.py` installs it as such, including an auto-generated `console_script` wrapper.
- `--dry-run` output now shows an actual docker command-line.
- Only pass `--tty` to docker if scuba’s stdout is a TTY.

7.22.3 Fixed

- Better handle empty `.scuba.yml` and other YAML-related errors
- Fix numerous bugs when running under Python 3

7.23 1.4.0 - 2016-01-08

7.23.1 Added

- Added `--verbose` and `--dry-run` options

7.23.2 Removed

- `umask` is no longer set in the container. (See [#24](#))

7.23.3 Fixed

- Problems introduced in v1.3.0 with Ctrl+C in images are fixed. The user command now runs as PID 1 again, as there is no more `.scubainit` script.

7.24 1.3.0 - 2016-01-07

7.24.1 Added

- Set `umask` in container to the same as the host (local Docker only)

7.24.2 Changed

- Change working directory from `/build` to `/scubaroot`
- Use `.scubainit` script to create `scubauser` user/group at container startup. This avoids the oddity of running as a uid not listed in `/etc/passwd`, avoiding various bugs (see [issue 11](#)). (local Docker only)

7.25 1.2.0 - 2015-12-27

7.25.1 Added

- Search up the directory hierarchy for `.scuba.yml`; this allows invoking `scuba` from a project subdirectory.
- Add `!from_yaml` support to YAML loading; this allows specifying image from an external YAML file (e.g. `.gitlab-ci.yml`).
- Add `CHANGELOG.md`

7.25.2 Changed

- Show better error message when docker cannot be executed

7.26 1.1.2 - 2015-12-22

7.26.1 Fixed

- Don't pass `--user` option when remote docker is being used

7.27 1.1.1 - 2015-12-22

7.27.1 Fixed

- Fix bug when `aliases` is not found in `.scuba.yml`

7.28 1.1.0 - 2015-12-20

7.28.1 Added

- Support for Bash-like aliases, specified in `.scuba.yml`

7.29 1.0.0 - 2015-12-18

7.29.1 Removed

- Remove the `command` node from `.scuba.yml` spec; it limits the usefulness of scuba by limiting the user to one command. Now command is specified on command line after scuba.

7.29.2 Added

- Argument parsing to scuba (`-v` for version)
- Check for and reject extraneous nodes in `.scuba.yml`

7.30 0.1.0 - 2015-12-09

First versioned release

CONTRIBUTING GUIDE

This file is incomplete. Feel free to open an issue if there is missing information you desire.

8.1 Code Format

Scuba is compliant with the [Black](#) code style. Code format in PRs is verified by a GitHub action.

To check code formatting:

```
$ ./code_format.py
```

To fix code formatting:

```
$ ./code_format.py --fix
```