Gerrie Documentation

Release 0.2

Andy Grunwald

December 04, 2014

Contents

Contents:

CHAPTER 1

Introduction

Gerrie is a data and information crawler for *Gerrit*, a code review system developed by Google.

Gerrie uses the SSH and REST-APIs offered by *Gerrit* to transform the data from Gerrit into a RDBMS. Currently only MySQL is supported. After the transformation the data can be used to start simple queries or complex analysis. One usecase is to analyze communites which use *Gerrit* like TYPO3, Wikimedia, Android, Qt, Eclipse and many more.

- Website: andygrunwald.github.io/Gerrie
- Source code: Gerrie @ GitHub
- Documentation: Gerrie @ Read the Docs

1.1 Features

- Full imports
- Incremental imports
- Full support of SSH API
- Command line interface
- MySQL as storage backend
- Debugging functionality
- Logging functionality
- Full documented

1.2 Read further

If you want to get start really quick please have a look at the *getting started guide*.

When you want to run this in a more proven environment please have a look at the *installation* and *configuration* chapter.

The *commands section* will explain the functionality of all commands implemented in Gerrie.

As a business analyst / data science engineer / math or numbers lover you can retrieve data and get an understanding of the database structure in the *database chapter*.

You want to contribute? Great! Get more information in the *contribution chapter*.

CHAPTER 2

Getting Started

This is a quick getting started guide. It will helps you to get the first try of Gerrie up and running.

Note: If you encounter errors or bugs during the getting started guide, don‘t give up! Open an issue or read about the details in the *Installation*, *Configuration*, *Commands* or *Database* chapter.

Download application and install dependencies:

```
$ git clone https://github.com/andygrunwald/Gerrie.git .
$ composer install
```

Copy config file and adjust configuration (*Database*, *SSH*, *Gerrit*):

```
$ cp Config.yml.dist Config.yml
$ vim Config.yml
```

A minimalistic configuration for the *TYPO3 Gerrit instance* with the user *max.mustermann* can look like:

```
Database:
  Host: 127.0.0.1
  Username: root
  Password:
  Port: 3306
  Name: gerrie

SSH:
  KeyFile: /Users/max/.ssh/id_rsa_gerrie

Gerrit:
  TYPO3:
    - ssh://max.mustermann@review.typo3.org:29418/
```

Create a new database in your database with name *gerrie* and setup database scheme:

```
$ mysql -u root -e "CREATE DATABASE gerrie;"
$ ./gerrie gerrie:setup-database --config-file="./Config.yml"
```

Create an account (e.g. *max.mustermann*) in the Gerrit instance you want to crawl (e.g. *review.typo3.org:29418*), add your SSH public key to the Gerrit instance and execute the *gerrie:check* command to check your environment:

```
$ ./gerrie gerrie:check --config-file="./Config.yml"
```

Note: **Important:** If your SSH key is protected by a passphrase this check will ask you to enter your passphrase

to use the private key for this connection. Gerrie does not save or transfer this passphrase to any foreign server. The private key is only necessary to authenticate against the Gerrit instance.

If everything is fine start crawling:

```
$ ./gerrie gerrie:crawl --config-file="./Config.yml"
```

Now the crawler starts and is doing its job :beer:

You reading can continue in the documentation in the chapters *Installation*, *Configuration*, *Commands*, *Database* or *Contributing*.

Note: Please note that we currently only support SSH and MySQL. We are open for changes and contributions. Feel free to push this product forward or get in contact with us.

CHAPTER 3

Installation

3.1 Prerequisites

3.1.1 tl;dr

- **PHP >= 5.4.0**
  - PHP Extension PDO
  - PHP Extension pdo_mysql
  - PHP Extension curl
- Composer
- MySQL >= v5.1
- SSH

3.1.2 PHP

Gerrie is written in PHP. To use Gerrie the PHP interpreter engine is necessary. Please install PHP. The installed version must be **>= 5.4.0**. See PHP Download and PHP Installation and Configuration.

If you install PHP from source please ensure that the PHP Extensions PDO, pdo_mysql and curl are also available. This extensions are part of the standard edition. If you install PHP by a standard package manager like apt you are on a save way :)

3.1.3 Composer

Gerrie include several 3rd party libraries. To manage this dependencies we use the defacto standard tool Composer for dependency management. Please install Composer. In the documentation you can find instructions to install it global or local. A **specific version is not necessary**. See Getting Started at Composer documentation.

3.1.4 MySQL

Gerrie uses the MySQL database as storage backend. Please install MySQL. The MySQL Community Edition is completely enough to fit Gerries need. The installed version must be **>= v5.1**. See MySQL Community Downloads.

3.1.5 SSH

Gerrie make use of the SSH API of Gerrit. To receive data via SSH the SSH client is necessary. Please install SSH. A **specific version is not necessary**. Most (or every) Linux / Unix distribution got SSH already installed.

3.2 Main instructions

3.2.1 Get the source

You can choose which version of Gerrie you want to install:

- the master branch
- a stable release

The difference between this two versions are:

- The master branch can be unstable, because this is the main development line
- The master branch can be ahead with new features and fixed bugs
- The latest stable release is stable and tested
- The latest stable release can be lack in features

To install the master branch via git just clone the source code:

```
$ git clone https://github.com/andygrunwald/Gerrie.git Gerrie
```

An alternative is to download the master branch as zip archive and extract it:

```
$ wget https://github.com/andygrunwald/Gerrie/archive/master.zip -O Gerrie.zip
$ unzip Gerrie.zip
$ mv Gerrie-master Gerrie
```

3.2.2 Install dependencies

Gerrie relies on several 3rd party libraries to speedup the development, make use of proven source code and avoid to reinvent the wheel.

```
$ cd Gerrie
$ composer install
```

3.2.3 Configure the application

Gerrie can be configured by a configuration file, options and arguments or both. The easiest solution is to copy the `Config.yml.dist` file to another location and adjust the settings in the self documented configuration file. This file will be added by the ‘‘–config-file‘‘option.

If you want to learn more about configuration in Gerrie or how to apply options and arguments *see the Configuration chapter*.

3.2.4 Execute the gerrie:check command

Gerrie got a build in command to check if your environment and configuration is working correctly. This command is named *gerrie:check*.

There are several ways how to execute the check command. The way depends on your preference. If you prefer a configuration file which contains all settings (see "Configure the application") then call:

```
$ ./gerrie gerrie:check --config-file="/Path/To/Config.yml"
```

If you prefer all settings passed as arguments to Gerrie this will be no problem. This command accepts many options and arguments. Get an overview with

```
$ ./gerrie gerrie:check --help
```

Here is an example call with using options and arguments instead of an configuration file plus a connection check for the Gerrit instance of the TYPO3 project.

```
$ ./gerrie gerrie:check --database-host="127.0.0.1" --database-user="gerrie" \
                        --database-pass="secret" --database-port=3306  \
                        --database-name="gerrie"  \
                        --ssh-key="/Users/max/.ssh/id_rsa_gerrie"  \
                        ssh://max@review.typo3.org:29418/
```

If everything works fine you will see red errors. If you got one or more errors please have a look at the *commands *gerrie:check* chapter*. There you can find a detailed description of the errors and hints how to fix them.

3.2.5 Run Gerrie, run!

If the *gerrie:check* went well, let Gerrie run. You have to know *Gerrie* loves crawling Gerrits :)

The main command of *Gerrie* is `gerrie:crawl`. Just execute it. It supports the same options and arguments as the `gerrie:check` command.

Without configuration file:

```
$ ./gerrie gerrie:crawl --database-host="127.0.0.1" --database-user="gerrie" \
                        --database-pass="secret" --database-port=3306  \
                        --database-name="gerrie"  \
                        --ssh-key="/Users/max/.ssh/id_rsa_gerrie"  \
                        ssh://max@review.typo3.org:29418/
```

or with configuration file:

```
$ ./gerrie gerrie:crawl --config-file="/Path/To/Config.yml"
```

or with both:

```
$ ./gerrie gerrie:crawl --config-file="/Path/To/Config.yml" --database-host="127.0.0.1" \
                        --database-user="gerrie" --database-name="gerrie" \
                        ssh://max@review.typo3.org:29418/
```

CHAPTER 4

Configuration

Gerrie supports two ways to pass a configuration in: **Options + Arguments** and **configuration file**. There is no need for a configuration file. All settings can be passed by options and arguments.

Advantage of using a configuration file:

- Shorter commands
- Simple versioning of configuration file

Advantage of using options + arguments:

- More flexible, because you don‘t need to modify a file

Next to the 0 and 1 solution you can combine both worlds. You can add a configuration file by option `--config-file` and overwrite attributes from the configuration file by options added to the command. The options got a **higher** priority as the attributes from the configuration file. The arguments will be **merged** with the Gerrit instances configured in the configuration file.

Example configuration file:

```
Database:
    Host: 127.0.0.1
    Username: root
    Password:
    Port: 3306
    Name: gerrit

SSH:
    KeyFile: /Users/agrunwald/.ssh/id_rsa_gerrie

Gerrit:
    TYPO3:
        - ssh://max@review.typo3.org:29418/
        - { Instance: ssh://max.mustermann@review.typo3.org:29418/, KeyFile: /Users/max/.ssh/id_rsa_
```

Example command:

```
$ ./gerrie gerrie:crawl --config-file=/path/to/example-conf.yml \
                        --database-host="192.168.1.10" -u="operator" \
                        https://max.mustermann:password@gerrit.wikimedia.org/ \
                        https://max:secret@android-review.googlesource.com/
```

In this example Gerrie will use:

- Database hostname: *192.168.1.10*

- Database username: *operator*
- Database password:
- Database port: *3306*
- Database name: *gerrit*
- ...
- **Instances:**
  - ssh://max@review.typo3.org:29418/
  - ssh://max.mustermann@review.typo3.org:29418/
  - https://max.mustermann:password@gerrit.wikimedia.org/
  - https://max:secret@android-review.googlesource.com/

4.1 Options

Gerrie supports a several options. Options are parameters prefixed by `--` or `-`. Example are in the long variant `--help` or in the short variant `-h`. Options can be accept a value (like `--config-file="..."`) or are standalone (like `--version`).

Note: Please have a look at the command you want to use first which options are supported. **Not all options are supported by all commands**. You can list options by command by `./gerrie gerrie:YOUR-COMMAND --help`. For available commands execute `./gerrie`.

Options will by added to the command like .. code:

```
$ ./gerrie gerrie:check --option1 --option2=value ...
```

Here you can find a list of all supported options.

4.2 Arguments

Next to options *Gerrie* supports arguments. Arguments are added at the end of the command separated by whitespace.

Note: Please have a look at the command you want to use first which arguments are supported. **Not all arguments are supported by all commands**. You can list options by command by `./gerrie gerrie:YOUR-COMMAND --help`. For available commands execute `./gerrie`.

Here you can find a list of all supported arguments.

Argument	Description
instances	List of instances to crawl separated by whitespace. You can add like many instances you want separated by whitespace Like “instance1 instance2 ... instanceN” Format: scheme://username[:password]@host[:port]/ Examples: - ssh://max.mustermann@review.typo3.org:29418/ - https://max.mustermann:password@gerrit.wikimedia.org/

4.3 Configuration file

The configuration file can be used to avoid long options and arguments. It can be located on the harddisk where *Gerrie* runs. The format of the configuration file is YAML. Ensure that you write the correct YAML syntax. YAML can be a little bit tricky when it comes to intention.

Note: In the root of *Gerrie* there is a *Config.yml.dist* which can be copied and used as a template for your configuration file. Don‘t forget to pass the path of the Gerrie.yml location as *-c* / *–config-file* option to the command.

If a attribute contains a ”.” this means that it will be a nested attribute. E.g. The attributes `Database.Host` and `Database.Username` will be in configuration file

```
Database:
    Host: 127.0.0.1
    Username: root
```

Here you can find a list of all supported configuration settings.

Attribute	Description
Database.Host	Name / IP of the host where the database is running.
Database.Username	Username to access the database.
Database.Password	Password to access the database.
Database.Port	Port where the database is listen.
Database.Name	Name of the database which should be used.
SSH.KeyFile	Path to SSH private key for authentication via SSH API.
Gerrit.Name1	Under the Gerrit namespace you can define several projects. The first level after `Gerrit` will be a name of the project. The name can be chosen by you and will be only used for internal. Internal use means for logging or store a relation between the name and n instances. The important info: The name can be chosen by you and you can use your wording. Example: Gerrit: TYPO3: ... Wikimedia: ...
Gerrit.NameN	As you can the in the example above you can define as many projects as you want.
Gerrit.Name1.0	The level below the project name is reserved for a list of instances per project. Instances can be - Gerrit server - Gerrit projects Instances can be added in several ways - a single url - a yaml array with a key `Instance` and a value as url - a yaml array with a key `Instance` and a value as url + a key `KeyFile` with a path to SSH key as a value The URLs are always in format `scheme://username[:password]@host[:port]/` The KeyFile will be used to connect to the related instance only and will overwrite the general KeyFile setting. A detailed example with possible formats is displayed below.
Gerrit.Name1.N	As you can the in the example above you can define as many instances per project as you want.

Note: Gerrit projects as an instance are not supported yet. This is planned for future versions.

Example showcase of five instances for the `TYPO3` and one for the `Wikimedia` project to display the possibility of `Gerrit.NameN.*`:

```
Gerrit:
  TYPO3:
    - Instance: ssh://max.mustermann@review.typo3.org:29418/
      KeyFile: /Users/max/.ssh/id_rsa

    - { Instance: ssh://max.mustermann@review.typo3.org:29418/, KeyFile: /Users/max/.ssh/id_rsa }

    - Instance: ssh://max.mustermann@review.typo3.org:29418/

    - { Instance: ssh://max.mustermann@review.typo3.org:29418/ }

    - ssh://max.mustermann@review.typo3.org:29418/

  # Second project
  Wikimedia:
    - https://max:password@gerrit.wikimedia.org/
```

CHAPTER 5

Commands

5.1 gerrie:crawl

The *gerrie:crawl* command is the main command of the *Gerrie* application. The main responsibility of this command is to receive the data from the Gerrit instance and transfer the data into a RDBMS. This happens in several steps:

1. Receive all information of configuration / options + arguments
2. Query the Gerrit instance for the / a project(s)
3. Transfer the project data into a unique format
4. Proceed (insert / update) project information
5. Query the Gerrit instance for Changesets + detailed information
6. Transfer the Changeset data into a unique format
7. Proceed (insert / update) changeset information

One requirement to execute *gerrie:crawl* is the necessary database structure. You can setup the table scheme with the command *gerrie:setup-database*. An alternative way can be the *–setup-database-tables* option of the *gerrie:crawl* command.

Note: At the moment Gerrie is only able to communicate with the SSH API of Gerrie. The support for the REST API is not build in yet in Gerrie.

Next to the described main logic there a several small features build in like

- debugging functionality to detect if every attribute which is received by the API is transformed to the unique format and no information is missing
- debugging functionality to detect if the project / instance is crawled the first time and only insert statements and not update statements will be executed
- logging to see what happens

5.2 gerrie:check

The *gerrie:check* command is useful to check your local / server environment if everything works fine with your current configuration. It accepts the same options and arguments as the *gerrie:crawl* command. With this it is easy to switch the command to check if everything if working.

The *gerrie:check* executes various checks to your environment. Current checks are:

- if the PHP extension *curl* is installed
- if the PHP extension *PDO* is installed
- if the PHP extension *pdo_mysql* is installed
- if SSH is installed and usable
- if the config file can be found
- if the config file is valid
- if Gerrie can connect to the database
- if Gerrie can connect the configured / passed Gerrit instances

During this checks several errors can occur. In the next sections we provide possible solution to fix your environment if a check failed.

5.2.1 PHP extension curl

If you see the message

```
PHP-Extension "curl" is not installed. Please install PHP-Extension "curl".
```

it seems to be that the curl extension is not installed or loaded in your environment. To check if *curl* is installed please list all available PHP modules with

```
$ php -m
```

and search for curl. If this is not in the list please have a detailed look in the Client URL Library @ PHP.net documentation. Specially the curl Installation chapter might be useful in your case.

5.2.2 PHP extensions PDO and pdo_mysql

If you see one of these messages

```
PHP-Extensions "PDO" and "pdo_mysql" are not installed. Please install both.
PHP-Extension "PDO" (v1.0.4dev) is installed, but "pdo_mysql" not. Please install it.
```

it seems to be that the PDO or pdo_mysql extension is not installed or loaded in your environment. With the following commands you can check if the module(s) are loaded and in which version they are available.

```
$ php -m
$ php -r 'var_dump(phpversion("PDO"));'
$ php -r 'var_dump(phpversion("pdo_mysql"));'
```

If this leads in a negative check it would make sense to have a look at the PHP Data Objects @ PHP.net documentation. Specially the PDO Installation chapter might be useful in your case. If everything is fine with PDO, but you got problems with pdo_mysql have a look at the pdo_mysql @ PHP.net documentation.

5.2.3 SSH

If you see the message

```
"ssh" is not installed. Please install "ssh".
```

it seems that the ssh executable can't be found or used. SSH must be executable to make use of the SSH API by Gerrit. If you need more information about SSH please have a look at OpenSSH.

You can test your SSH by

```
$ ssh -V
```

5.2.4 Config file location

The config file location got two different error messages. If you see the message

```
Config file "X" was not found.  Please provide the correct path or all settings via command options.
```

the config file can't be found. You can just copy the *Config.yml.dist* in the same folder, adjust it and pass it via *-c* / *–config-file* option. This would fix the problem:

```
$ cd /path/to/Gerrie
$ cp Config.yml.dist Config.yml
$ ./gerrie [...] --config-file="Config.yml"
```

With this you can put the config file wherever you want.

Note: The configuration file is not required. You can pass all settings by options and arguments. If this check fail ensure that you will use the options + arguments.

If you see the message

```
Config file "X" was found, but is not readable. Please change ownerships or all settings via command
```

then your configuration file was found, but is not readable by the user which executes the Gerrie application. Please adjust the access rights. Maybe Chmod and Chown can help you.

5.2.5 Config file validation

If you see a message like

```
The configuration is not complete. Missing keys are X. Please provide them as command options.
```

the configuration file is not complete. There are the mentioned settings missing. If you don't know which keys need to be in the config file, please have a look at the self documented *Config.yml* in the root directory of Gerrie. The *Configuration chapter* will list all available settings as well.

Note: The configuration file is not required. You can pass all settings by options and arguments. If this check fail ensure that you will use the options + arguments.

5.2.6 Database connection

If you see a message like

```
Database connection to host "120.0.0.1" works not as expected. Please check your credentials or setup
```

Gerrie can't build a database connection. A database connection is required to use Gerrie. To check if your database is working you can try to connect with the same credentials via commandline:

```
$ mysql -h 127.0.0.1 -uUSER -p
$ # enter password here
$ mysql> USE DATABASENAME;
$ mysql> SHOW TABLES;
```

Note: Only MySQL is supported.

5.2.7 Gerrit instance connection

Depending on your configuration you will use the SSH or HTTP / REST API by Gerrit. Both connection kinds can fail and will output a error message like

```
Connection to Gerrit "review.typo3.org" via SSH-DataService was not successful. Please check your cre
```

Please read further to fight against your issue.

Connection via SSH

The SSH API is a little bit tricky.

At first the Gerrit instance must support access by SSH. Instances like TYPO3 or Wikimedia does this. Instances like Android (which are hosted at googlesource) does not. They only support HTTPS.

One requirement is that you got a user account at this instance and your SSH public key was added in Gerrit at *Settings > SSH Public Keys*. After this you can test your command with

```
$ ssh -i /Path/To/Your/Private/.ssh/key -p 29418 USERNAME@HOST gerrit version
# e.g.
$ ssh -i /Users/max/.ssh/id_rsa_gerrie -p 29418 max.musterman@review.typo3.org gerrit version
```

A valid response should be

```
gerrit version 2.9.1
```

If you see something like “Access denied” please check your private / public key pair.

Connection via HTTP(S)

The HTTP(S) API is a little bit more easier to use than the SSH API. Mostly every current version of Gerrit supports the REST-API.

Note: The HTTP(S) API is not fully supported by Gerrie. This is planned for future versions of Gerrie.

There are two ways to test the REST-API: With and without authentification. At first be sure that this works without authentification. This is easy and you can just request a special url with curl like

```
$ curl SCHEME://HOST/config/server/version
# e.g.
$ curl https://review.typo3.org/config/server/version
```

A valid response should be

```
)]}'
"2.9.1"
```

Next step would be to check the access via REST API with your user credentials. You can do this via curl as well:

```
$ curl --user USERNAME:PASSWORD SCHEME://HOST/a/accounts/self/username
# e.g.
$ curl --user max.mustermann:mypassword https://review.typo3.org/a/accounts/self/username
```

A valid response should be

```
)]}'
"max.mustermann"
```

If you got a response like

```
Unauthorized
```

please check your username and password at the Gerrit instance.

Note: To crawl a Gerrit instance a authentification is not necessary for the REST-API. This depends on your user account. For example some instances give logged in users a higher API ratio or more rights to see more projects.

5.3 gerrie:setup-database

The *gerrie:setup-database* command is responsible to setup the database scheme (tables only) for the *gerrie:crawl* command. The *gerrie:setup-database* command won‘t create the database it selfs. The database have to already exist.

If the database contains tables the command won‘t overwrite something. The command checks if a table with the same name already exists (this is done by *SHOW TABLES LIKE ...*). If yes, the command does nothing and will execute the same procedure for the next table. If the requested table does not exist it will be created (this is done by *CREATE TABLE ...*).

What does this mean? This means that if you will upgrade from an old version of Gerrie to a newer one and you know that there were database schema changes that this changes won‘t be applied to your already existing scheme. Database scheme changes has to be applied by a different way.

In the normal world you can apply the *gerrie:setup-database* to every existing database which contains various tables already. All tables which will be created by *Gerrie* as prefixed by *gerrie_*. If you do not get any tables which got the prefix *gerrie_* you can just apply this command to your database.

CHAPTER 6

Database

6.1 Supported databases

Currently *Gerrie* supports only MySQL.

Note: You want to support another database like PostgreSQL or a NoSQL database? Feel free to *contribute*. This is _very_ welcome!

6.2 Setup the schema

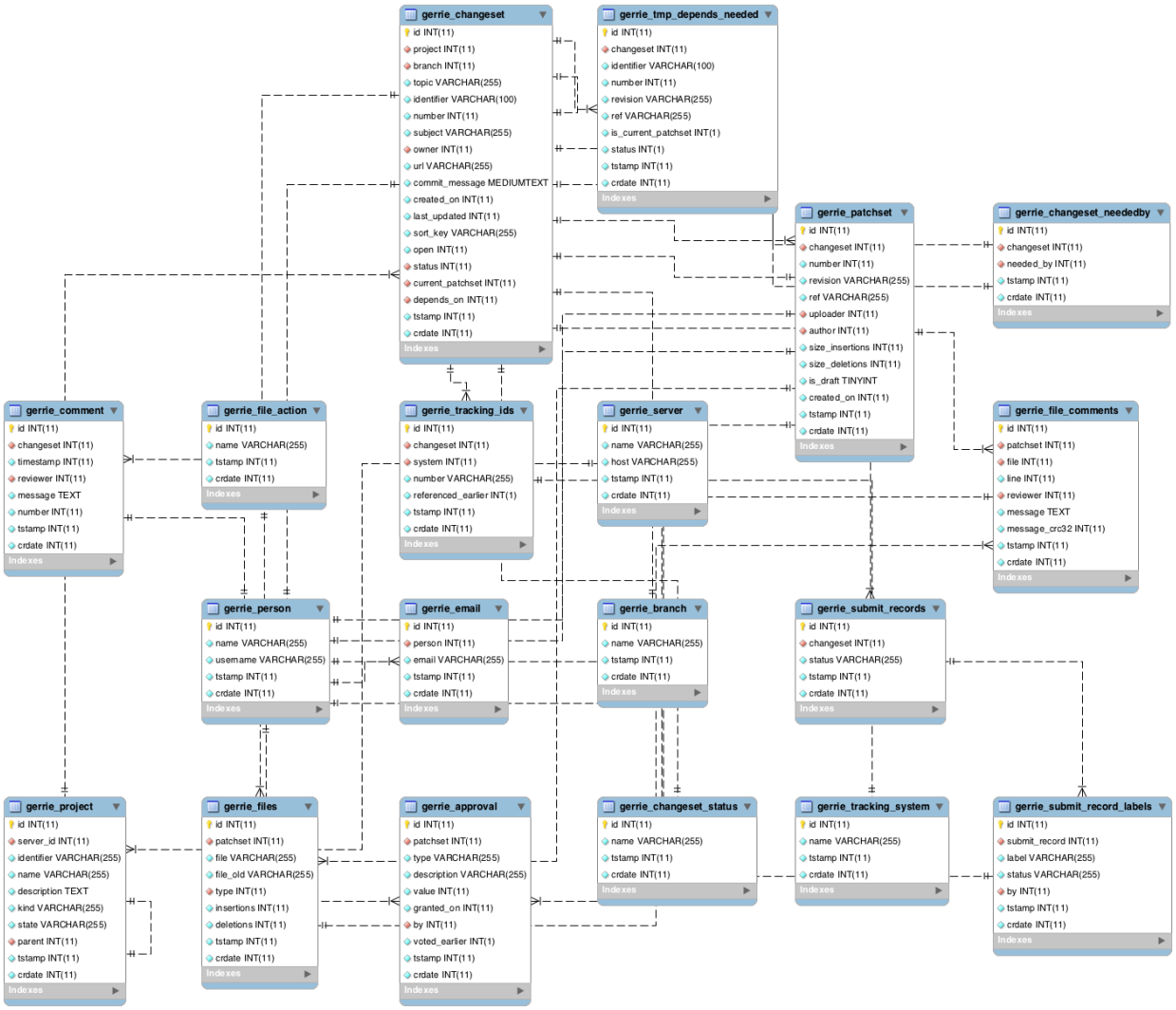
To setup the schema the database itselfs has to be exists. Please create this database before using *Gerrie*. A command can look like

```
CREATE DATABASE gerrie;
```

Based upon this you can create the database schema. The *gerrie:setup-database command* will help you to setup the database schema.

6.3 Schema

To query the database and answer questions based on this data it is important that you will understand how the data is structured. MySQL is a typical RDBMS. So we use a typical table / column schema to reflect the complex data structure of Gerrit. Below you will find a visualisation of the database scheme as entity relationship model.



This entity relationship model was created with MySQLWorkbench. If you want to get a deeper look at this schema to investigate it, you can install MySQLWorkbench and load the source file eer-diagram.mwb.

CHAPTER 7

Contributing

7.1 Writing Code

7.1.1 General

Coding Style Guide

For convenience we follow the PSR-1 and PSR-2 coding style guides of PHP Framework Interop Group. Please be so nice to take care of this during code contribution (e.g. pull requests). To check your code against this standards you can use tools like PHP_CodeSniffer.

7.1.2 Pull requests

With Gerrit we follow the standard code contribution of the GitHub platform. This means:

1. **Fork the project into a personal username space and clone the repository.**

   ```
   $ git clone https://github.com/andygrunwald/Gerrie.git
   ```

2. **Create a new git branch for your change (bugfix, feature, improvement, etc.).**

   ```
   $ git checkout -b my-new-feature
   ```

3. **Make your changes in the codebase until your changes are working.**

   ```
   $ vim ./file
   $ # Hack hack hack
   ```

4. **Commit your changes into your local git repository.**

   ```
   $ git commit -am 'Add some feature'
   ```

5. **Push your new branch to your fork repository.**

   ```
   $ git push origin my-new-feature
   ```

6. **Visit the forked repository via the GitHub website and create the pull request based on your new branch.**

   ```
   $ # Ploepp (beer open)
   $ # gluck gluck gluck (beer drinking)
   ```

This are the necessary steps described in a really rough way. If you need more help the GitHub help pages are a a excellent source:

- Fork A Repo
- Creating a pull request
- Using pull requests
- Syncing a fork
- Merging an upstream repository into your fork
- Configuring a remote for a fork

7.1.3 Testing

PHPUnit

Gerrie uses PHPUnit to create unit and integration tests. The tests are located in the `tests` folder. To execute the unit tests ensure that you have installed all development dependencies via `--dev` and start PHPUnit:

```
$ composer install --dev
$ # Without code coverage generation
$ ./vendor/bin/phpunit --coverage-clover=coverage.clover
$ # With code coverage generation
$ ./vendor/bin/phpunit
```

Note: To generate the code coverage you need the PHP Extension `xDebug` installed.

To create mock objects we use the standard functionality of PHPUnit.

7.1.4 Quality services

Travis CI

Travis CI is a free hosted Continuous Integration Platform for Open Source projects. We make us of this service to execute our tests continuous. One of the biggest advantages is that all pull request will be checked with Travis CI as well. So if you want to contribute please do not fear to break something. Every pull request you create will be checked and you will be notified if something go wrong. So just try it :)

See andygrunwald/Gerrie @ Travis CI.

Scrutinizer

Scrutinizer is a free hosted Continuous inspection Platform for Open Source projects. This service executes several checks for us like * checking the coding styleguide for us * observe the code documentation about possible bugs in return values * determine a quality score for *Gerrie* * and adds small tipps of how to improve the code quality of the software

As a small additional feature we push the generated code coverage from our unit tests from Travis CI to Scrutinizer. With this Scrutinizer can determine the overall code coverage for us.

See andygrunwald/Gerrie @ Scrutinizer.

7.2 Writing Documentation

Documentation is really important. Especially for an Open Source project where no one is payed to work with (maybe legacy) software. Documentation helps user * to understand how to use software * what are basic concepts of the tool * to follow the original thoughts of the author * to start contributing to the tool * understand how they can be involved * and many things more.

This are the reasons why documentation is important. We at Gerrie thinks that to write documentation must be easy. That is the reason why we store documentation as **plain text next to the source code**. You can find our documentation in the docs/ directory.

Learn more about how we write and where we host and generate our documentation.

7.2.1 reStructuredText

We write our documentation in reStructuredText:

> reStructuredText is a file format for textual data used primarily [..] for technical documentation.

reStructuredText is very similar to MarkDown. ReStructuredText is easy to write and really powerful. You can convert this to several formats like HTML, ePub or PDF.

To learn the reStructuredText syntax have a look at reStructuredText @ Wikipedia or Quick reStructuredText of docutils.

7.2.2 Read the Docs

We use the service of Read the docs for hosting and generating our documentation. Read the Docs makes it really easy to write, build and manage documentations of Open Source projects. The maintenance is less and the author can focus on the relevant topics instead of maintaining infrastructure.

To start and use Read the Docs and to compile and test the documentation you have to install a few Python tools. For example Read the Docs uses Sphinx to render the documentation. Details about the necessary software can be found in the Getting started @ Read the docs.

To render our documentation just checkout the source code from GitHub, compile the documentation and open it:

```
$ git clone https://github.com/andygrunwald/Gerrie.git Gerrie
$ cd Gerrie/docs
$ make html
$ open _build/html/index.html
```

After the new docs are pushed to the main Repository of GitHub, Read the Docs will render the new version and publish it to Gerrie @ Read the Docs.

7.3 Reporting Issues

It is important for an Open Source project to get feedback, bugs, feature requests, enhancement , tips, tricks, hints and more from users and contributers. We at Gerrie are using the standard issue tracker at andygrunwald/Gerrie from GitHub for this.

Please feel free to open an issue at our issue tracker to: * report a bug * suggest a feature request * notify us in case of typos * ask questions * give feedback (positive or negative) * share tips, tricks and hints * offer sponsoring * get in contact with the authors * suggest dates for beer drinking

We are waiting for your issue :)

7.4 Various

Contributing source code, documentation or issues are not everything. **There are several other ways to contribute to an open source project and this ways are important as well!** In this area we will describe some ways in the next sections.

7.4.1 Talk about Gerrie

As every other product / service word-of-mouth advertising is important. Feel free to talk about Gerrie if you are using it, you are happy because it solves your needs and you want to contribute. You can talk about it in thousand ways:

- Presentation at conferences
- Presentation at usergroups
- Write a blog post
- Write a post on a mailing list
- Mention it in academic researches / papers
- Chat with a friend or colleague during a beer
- you will find a way

If you talk about Gerrie in a public way we would **love** to hear about this activity. Would you be so nice to drop us a mail or a ticket / issue ? Thank you!

7.4.2 Sponsoring

If you think "Hey, Gerrie is a cool and useful project. It helps me, my team or my company a lot" it would be nice if you think about to contribute back with a small sponsorship. If we are talking about a sponsorship there are several different ways as well:

- **Financial**: You can sponsor money back to us e.g. to take a bill of a pizza, a beer or for a new feature.
- **Developer**: If you got a team of developer (e.g. you are an agency / company owner) we would love to see new contributer who will fix a bug or add a new feature.
- **Designer**: This is quite the same as for Developer, but for designer. Especially now, because we are *looking for a logo*

7.4.3 Contact

We love to get in contact with our users. To receive feedback (positive or negative) is important to learn how users will use this tool and how we can improve it further.

Do not hesitate to get in contact with us. You can ask as everything. Maybe you don't know where / how to start with? Or just want to share your opinion about Gerrie?

Start now to contact us by a ticket / issue or by email to the contributers. Thanks in advance.

7.5 Logo

Every product needs a logo. Even Gerrie. Sadly the current authors are not very good in designing a meaningful logo. This is the reason why we hope that a designer got some time to create a new shiny logo :)

This logo can be used to *talk about it at conferences or usergroups* or to create a recognition value.

We don‘t get a special briefing. The designer got free space to design it. In short what we are doing:

> A crawler for Googles code review system “Gerrit”.
>
> Gerrie uses standardized API (like SSH or REST) offered by Gerrit to transform the complex data structure from Gerrit into a relational database like MySQL.

gerrit, the open source project got already a logo: Diffy - The Kung Fu Review Cuckoo

Feel free. We would like to receive a nice suggestion from you. Sadly we do not get a budget for it, because we do not earn money with Gerrie. Everything what we can offer is fame. You will be mentioned everywhere we will publish this logo.

CHAPTER 8

License

This project is released under the terms of the MIT license.

```
The MIT License (MIT)

Copyright (c) 2014 Andreas Grunwald and contributors.

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.
```

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*