gitexplorer Documentation

Release 0.1

Peer Wagner

Sep 26, 2017

Contents:

This project is intended to be a tool to extract basic information from any accessible git repository, make appealing visualizations like the GitHub graphs and therefore make exploration of repositories as easy as possible.

Being a fairly new project neither all requirements are written nor are implementation details already clear. I will take the chance and document the process of architecture and design decisions. As an inspiration for the project I base on the great repositories hoxu/gitstats and adamtornhill/code-maat.

In the future the starting point for all interaction with the package gitexplorer will start with:

```
import gitexplorer as ge
```

So stay tuned ...

CHAPTER 1

The story behind gitexplorer or the What, Why and for Whom!

As a computer scientist working with many different kinds of software and in various teams, I was always curious on how to improve my work and the software I am working with. Almost naturally, as it comes to bigger applications and bigger teams, me and my team always used a version control system. Therefore I was always interested in how to use those systems to get information about the past; sometimes by reverting bugs some of us introduced, to retrace ideas or to get an overview on what is going on with our source code, not to say who is doing what and where are those changes located.

Reading a lot about refactoring (especially *[Torn15]*), I have an emerging interest in analyzing my repositories in a more structured way. Hopefully this will enable me to refactor the code which needs it most to simplify my daily work.

Being a Python programmer I searched the internet for a Python application fullfilling my wish for a way to analyze git repositories to get the information I was looking for. As I formerly worked with SVN and remembered the tool StatSVN which generated a lot of statistics about a repository, I immediately found GitStats. To my displeasure it had a strong coupling to gnuplot and after contacting the maintainer it was clear that the project was no longer under active development despite lots of pull requests to improve it. A look at the source code and a few modifications later I decided to start a project on my own. An application capable to generate all the statistics I wish for, combined with an appealing output.

Requirements

I formulated the following few non specific requirements for myself:

1. An application which can be used to analyze and visualize arbitrary git repositories. The data and visualization artifacts can then be used to get a deeper insight into usage and content of the analyzed repository.
2. The result of the analysis shall be persisted to be efficiently accessible for visualization and reevaluation as well as future extension.
3. Extending the application shall be possible by writing additional visualizations and/or additional data analysis which can result in extra information to be stored in conformity to #2.
4. The style of visualizations shall be easily changeable by programmers and non programmers to support separation of concerns.

5. The analysis of the repository shall be as effective as possible whereas the resulting information storage shall be partially updatable and upgradable.

Achivements & Goals

This documentation should be an up to date source of information which statistics are already available in the basic package and which are soon likely to be available.

Done

1. Total number of additions, deletions, lines and modifications per commit
2. Total number of additions, deletions, lines and modifications per commit, grouped by date
3. Number of commits grouped by iso day of the week
4. Number of commits grouped by hour of day
5. Authors and corresponding date of commits, additions, deletions and lines grouped by file path
6. Additions, deletions, lines and commits grouped by file path
7. Average number of additions, deletions, lines and modifications per commit
8. Average number of additions, deletions, lines and modifications grouped by date

To Be Done

1. List of file extensions
2. Total number of additions, deletions, lines and modifications per author
3. Average number of additions, deletions, lines and modifications per commit, grouped by author
4. Every other statistic limited to the last 30 days
5. Every other statistic limited to the last half year
6. Every other statistic limited to the last year
7. Total number of commits per file path
8. Author of the month rated by 'TBD'
9. Author of the year rated by 'TBD'
10. Every statistic respecting renames
11. Total number of additions, deletions, lines and modifications per extension
12. Size per file
13. Update frequency per file
14. Coupling of files
15. Python specific file statistics like cyclomatic complexity
16. Visualization of all statistics

CHAPTER 2

Building the architecture which meets the requirements

Where do we start? The first try of a design fulfilling some of the architectural requirements could look like visualized in *Figure 1*. A script will read out the git information and put it into a persistent storage. This storage will be read out from a script generating the visualization with respect to some configuration options.

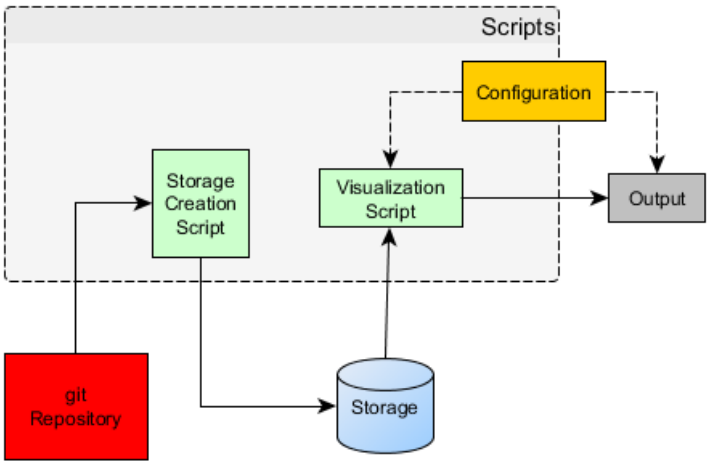

Fig. 2.1: **Figure 1:** Basic architecture for gitexplorer.

However if we think about supporting the extensibility requirement, it is clear that this architecture can be improved.

```
{"commit_hash": <commit_hash>,
 "author": <name>,
 "mail": <mail>,
 "date": <date>,
 "details": {
     "create": [{
         "file_path": <file_path>,
         "permission": <unix_file_permission>,
```

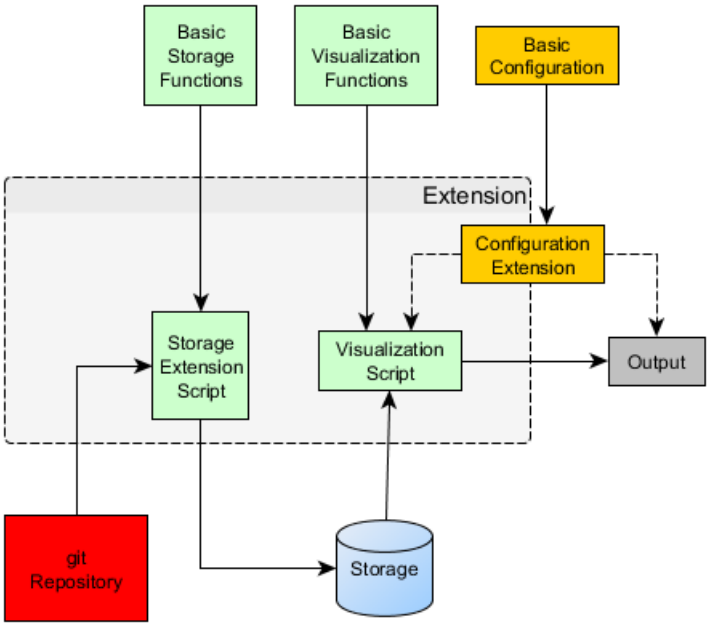

Fig. 2.2: **Figure 2:** Architecture for gitexplorer having extensibility as a basic design element.

```
        "extension": <.extension>}],
    "delete": [{
        "file_path": <file_path>,
        "permission": <unix_file_permission>}],
    "rename": [{
        "new_path": <file_path>,
        "extension": <.extension>,
        "old_path": <file_path>,
        "match": <match_percentage>}],
    "change": {
        <file_path>: {
            "old_permission": <unix_file_permission>,
            "new_permission": <unix_file_permission>}},
    "modifications": [{
        "file_path": <file_path>,
        "additions": <#additions>,
        "deletions": <#deletions>}]}}
```

CHAPTER 3

License

Project License

```
MIT License

Copyright (c) 2017 Peer Wagner

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.
```

Logo License

The “gitexplorer logo” is a derivative of the original “Git Logo” by Jason Long, used under Creative Commons Attribution 3.0 Unported License. “gitexplorer logo” itself is licensed under Creative Commons Attribution 3.0 Unported License by Peer Wagner.

CHAPTER 4

Developers

- Peer Wagner <wagnerpeer@gmail.com>

CHAPTER 5

Changes

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[Torn15] Tornhill, A. (2015). Your code as a Crime Scene - Use Forensic Techniques to Arrest Defects, Bottlenecks and Bad Design in Your Programs. Dallas, TX: The Pragmatic Bookshelf