Authorship Test

Ketlin de Mello Moreira David M. Rogers

June 25, 2021

1 Introduction

GitHub, GitLab, BitBucket, and other web-services provide a browser-based interface to interact with git repositories. These web-services are designed around a model-view architecture, where the webserver maintains an internal database (the model) that it presents to web users via html pages and JSON API[1] calls (the views).

This architecture can be a source of confusion for repository data-mining, because the web-service's database contains *different* information than the underlying git repository. 'Scanning' a web-service via making JSON API calls shows information retrieved from the web-service's database. As long as the web-service's database is in sync with the underlying git database, the scan results will contain a superset of repository and web activity. If the two are out of sync, the scan results can only be said to contain the state of the web database.

In order to understand how interactions with github change our view of their database, we have undertaken the following experiment. First, a test repository has been setup using a series of interactions with git and github. Next, several queries were run against the github JSON API using the py-github module (https://github.com/PyGithub/PyGithub).

This document describes the setup and scan phases of the experiment.

2 Application Overview

The "authorship-tests" is a testing repository used to determine the full extent of identification attributes available to contributor' authorship as well as to find any unexpected properties of the data. There are several contributions made to the repository like Pull Requests, Commits, Merges and Issue Reports which are integrated to fulfill the purpose of the API calls.

[1]JSON = 'JavaScript Object Notation', API='Application Programming Interface'

2.1 Testing Scope

In-Scope

Comprehensive authorship data available on the GitHub database including all the git files, logs, and interactions direct to the web interface.

Out of Scope

Authorship interactions on GitHub database comprehending bug reports and denied pull requests.

3 Creation of the Test Repository

The "authorship-tests" repository was created using the "New Repository" mechanism from the GitHub website. A first contribution was made by the main author following the listed cmd.exe commands:

```
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

c:\User\authorship-tests> git init
c:\User\authorship-tests> git config user.name "First Last"
c:\User\authorship-tests> git config  user.email "example@email.com"
c:\User\authorship-tests> git add .
c:\User\authorship-tests> git commit -m "Comment"
c:\User\authorship-tests> git branch
c:\User\authorship-tests> git remote add origin https://github.com/
    bssw-psip/authorship-tests.git
c:\User\authorship-tests> git push -u origin master
```

Next, one of the authors in the 'bssw-psip' project (Reed Milewicz) forked the project to 'rmmilewi/authorship-tests', where he pushed a code update to the master branch and submitted a github pull-request. Ketlin de Mello merged this into the 'bssw-psip/authorship-tests' repository.

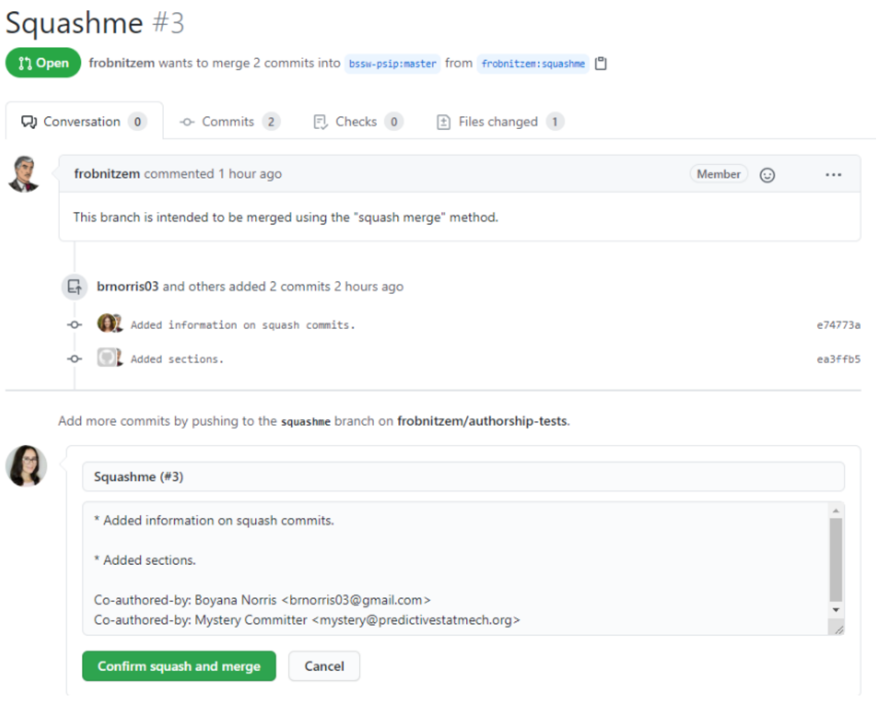
Then, another author in the 'bssw-psip' project (Benjamin Sims), proposed a change using the github web interface. This resulted in creation of a 'bhsims-patch-1' branch within the project. The pull request was closed without merging into the 'master' branch.

Then Elaine M. Raybourn added an issue to the 'bssw-psip/authorship-tests' repository using the web interface.

Next, David M. Rogers forked the project into 'frobnitzem/authorship-tests', merged Benjamin Sims' changes, and added two git-commits with fake author information.

This can be accomplished using the git command:
`git commit --author="Mystery Committer <mystery@predictivestatmech.org>"`. These changes were sent via pull-request to 'bssw-psip', where Ketlin de Mello merged them with the 'Squash and Merge' functionality.

Next, the master branch of the 'bssw-psip/authorship-tests' repository was manually reset back to the state before the squash-merge above. This was accomplished using `git reset --hard c4696796` and `git push -f origin master`.


Squashme #3

Open frobnitzem wants to merge 2 commits into bssw-psip:master from frobnitzem:squashme

frobnitzem commented 1 hour ago Member

This branch is intended to be merged using the "squash merge" method.

brnorris03 and others added 2 commits 2 hours ago

```
Added information on squash commits.    e74773a
Added sections.                         ea3ffb5
```

Add more commits by pushing to the squashme branch on frobnitzem/authorship-tests.

Squashme (#3)

* Added information on squash commits.

* Added sections.

Co-authored-by: Boyana Norris <brnorris03@gmail.com>
Co-authored-by: Mystery Committer <mystery@predictivestatmech.org>

Figure 1: Squash and merge feature of GitHub

Next, David M. Rogers prepared ‘frobnitzem/authorship-tests’ as before to add two git-commits with fake author information. This time, Benjamin Sims’ commits were not included.

Finally, these changes were sent via pull-request to ‘bssw-psip’, where Ketlin de Mello merged them with the ‘Squash and Merge’ functionality.

4 GitHub API Query Results

Each of the subsections below describes a query run against the final state of the ‘bssw-psip//authorship-tests’ repository, along with the results returned.

4.1 Contributor by Issue

The following python function exemplifies the data collected for contributors by issues. Issue raisers are not identified on GitHub web interface as contributors or authors of a repository. To gather their contributions it was needed to query for authors or actor of each event.

```
def getcontributorByIssues():
    issueRaiser = PrettyTable()
    issueRaiser.field_names = ["Issue Raiser"]

    issues = repository.get_issues()
    for issue in issues:
        issueRaiser.add_row([issue.user.name])
    return issueRaiser
```

```
+--------------------+
|    Issue Raiser    |
+--------------------+
| Elaine M. Raybourn |
+--------------------+
```

4.2 Commit Author

The following python function exemplifies the data collected from all pull requests, and given each request it returns the authorship of commits. Once a Pull Request is squashed and merged, or a commit is deleted from GitHub the author’s names are still shown on the GitHub web interface as contributors if the email used at the time of commit can be associated with a GitHub account. As GitHub web interface do not keep track of all commit history, it does not list information associated with deleted or merged commits. In order to identify the entire history of authorship, it is necessary to query the GitHub web API that contains all the data from git files and their logs, plus all the data history of interactions with the web interface.

```
def getContributorByCommitsByPullRequest():
    commitAuthor = PrettyTable()
    commitAuthor.field_names = ["Commit_Author"]
    pullRequests = repository.get_pulls("all")
    for pr in pullRequests:
        for comm in pr.get_commits():
            commitAuthor.add_row([comm.commit.author.name])
    return commitAuthor
```

```
+--------------------+
|    Commit Author   |
+--------------------+
|    Boyana Norris   |
| Mystery Committer  |
|    Boyana Norris   |
| Mystery Committer  |
|    Benjamin Sims   |
|    Reed Milewicz   |
+--------------------+
------------------------------------------------------------------
 PR Name: Added a new branch
 Commit Name: Please merge this using the "squash merge"
    feature.
 Commit ID:  d143f01b3a558b0b315af981b324654fbec47044
    Commiter Name: David M. Rogers
    Commiter Login: frobnitzem
    Commit Author Login: NamedUser(login="brnorris03")
    Commit Author Name: Boyana Norris
    Commit Author email: brnorris03@gmail.com
 Commit ID:  8efa2dd581ca44cd2dbfeddc6afdcd575ebb5f5b
    Commiter Name: David M. Rogers
    Commiter Login: frobnitzem
    Commit Author Login: None
    Commit Author Name: Mystery Committer
    Commit Author email: mystery@predictivestatmech.org
------------------------------------------------------------------
 PR Name: Squashme
 Commit Name: This branch is intended to be merged using
    the "squash merge" method.
 Commit ID:  e74773a224a6a84286513e34ef74a6929f5c0e60
    Commiter Name: David M. Rogers
    Commiter Login: frobnitzem
    Commit Author Login: NamedUser(login="brnorris03")
    Commit Author Name: Boyana Norris
    Commit Author email: brnorris03@gmail.com
 Commit ID:  ea3ffb5351ac0c651a3d6f59a890ec8016a0f6b6
```

```
    Commiter Name: David M. Rogers
    Commiter Login: frobnitzem
    Commit Author Login: None
    Commit Author Name: Mystery Committer
    Commit Author email: mystery@predictivestatmech.org
--------------------------------------------------------------------------------
```

4.3 Contributor by Event

The following python function exemplifies the data collected from events. When querying the web API for contributors by event, the most relevant and comprehensive data for authors and contributors can be identified.

```
def getContributorByEvents():
    eventContributor = PrettyTable()
    eventContributor.field_names = ["Contributor by Event"]
    events = repository.get_events()
    for event in events.get_page(0):
        if event.actor.name == None:
            eventContributor.add_row([event.actor.login])
        else:
            eventContributor.add_row([event.actor.name])
    return eventContributor
```

```
+----------------------+
| Contributor by Event |
+----------------------+
|     Ketlin Mello     |
|   David M. Rogers    |
|  Elaine M. Raybourn  |
|    Benjamin Sims     |
|    Reed Milewicz     |
+----------------------+
```

4.4 Stats Contributor

The following python function exemplifies the steps to collect data for stats contributors.

```
def getStatsContributor():
    stastsContributor = PrettyTable()
    stastsContributor.field_names = ["Stats Contributor"]
    stats = repository.get_stats_contributors()
    for stat in range(len(stats)):
        if stats[stat].author.name == None:
            stastsContributor.add_row([stats[stat].author.login])
        else:
            stastsContributor.add_row([stats[stat].author.name])
    return stastsContributor
```

```
+----------------------+
|  Stats Contributor   |
+----------------------+
|    Reed Milewicz     |
|     Ketlin Mello     |
|   David M. Rogers    |
+----------------------+
```

4.5 Contributor

The following python function exemplifies the steps to collect data for contributors.

```
def getContributor():
    cont = PrettyTable()
    cont.field_names = ["Contributors"]
    contributors = repository.get_contributors()
    for contributor in contributors:
        if contributor.name == None:
            cont.add_row([contributor.login])
        else:
            cont.add_row([contributor.name])
    return cont
```

```
+----------------------+
|     Contributor      |
+----------------------+
|    Reed Milewicz     |
|     Ketlin Mello     |
|   David M. Rogers    |
+----------------------+
```

4.6 Collaborators

The following python function exemplifies the steps to collect data for collaborators.

```
def getCollaborators():
    collab = PrettyTable()
    collab.field_names = ["Collaborators"]
```

```
collaborators = repository.get_collaborators()
for collaborator in collaborators:
    if collaborator.name == None:
        collab.add_row([collaborator.login])
    else:
        collab.add_row([collaborator.name])
return collab
```

```
+---------------------------+
|       Collaborators       |
+---------------------------+
| Elsa Gonsiorowski, PhD    |
|      David M. Rogers      |
|     James Willenbring     |
|        Greg Watson        |
|       Reed Milewicz       |
|      Han Yong Wunrow      |
|     Elaine M. Raybourn    |
|       Benjamin Sims       |
|         jmox0351          |
|        Ketlin Mello       |
+---------------------------+
```

5 Conclusion

This testing confirms that GitHub's API returns much more data than is present within the underlying git history. Thus, the term 'repository' is ambiguous on its own. The context needs to make it clear whether the repository refers to a git repository (a list of objects, commits, and references to commits), or to a GitHub repository, which is a web database that provides a layer of data on top of an underlying git repository.

Authorship can be quantified in many different ways. The information gained above allows us to understand what kinds of authorship are seen by each GitHub JSON API query.

In particular,

- **Collaborators** has nothing to do with authorship, but explains github access permissions.
- **get_contributor** and get_stats_contributors show the github logins responsible for interacting with the pull-request functionality (e.g. committers, not commit-author).
- **Commit Authors** are what we would typically refer to as the authors of source code. There is no direct API call to retrieve them.

- **issues** are a separate category of authorship that captures interaction through github's web API. NOTE: if you call the get_issues function and specifically ask for both OPEN and CLOSED issues, you would also see Benjamin Sims in the list of issue authors. This is because a pull-request is counted as an issue by github.

This test showed that **Commit Authors** can be retrieved by querying github for its history of commits. Github retains a memory of commits that are deleted after forced updates to the repository, so that the commit history will differ from the commits viewed in the underlying git repository. Commits that have been merged using the squash-merge feature preserve authorship by adding a special acknowledgment line to the commit message.