A Multi-level Dataset of Linux Kernel Patchwork

Yulin Xu
Minghui Zhou*
School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education, China
kylinxyl@pku.edu.cn
zhmh@pku.edu.cn

ABSTRACT

In many open source software projects (e.g., the Linux kernel), people contribute by sending code patches to the community. The community evaluates these contributions and decides whether to integrate the changes. To improve the efficiency of code contributions, substantial effort has been devoted to analyzing how patches are submitted and processed. Patch data are critical for this type of analysis, while retrieving and cleaning the data is a non-trivial job. To facilitate these studies, we share a multi-level dataset of a Linux kernel patchwork covering a nine-year history of patches and related discussion recorded by the Linux kernel mailing list (LKML). The data and scripts are provided at: https://zenodo.org/record/1165576

ACM Reference Format:
Yulin Xu and Minghui Zhou. 2018. A Multi-level Dataset of Linux Kernel Patchwork. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196475

1 INTRODUCTION

Open source software (OSS) projects often rely on code contributions from a wide variety of developers to extend the capabilities of their software. Sending a patch through a mailing list is a primary means of contributing to many traditional OSS projects[1] (e.g., the Linux kernel). The project members receive the patches, review them, and decide whether to accept them.

Many researchers have investigated patch data produced in open source projects to understand how the software is developed [1] as well as to investigate the possible problems in the development process [4, 8, 10] and to devise methods or tools to improve the efficiency of development [3, 6, 7]. The Linux kernel is a typical community that uses patches to accept contributions. Linux has had a profound impact on computing and society in general and has attracted considerable research interest with respect to its patchwork. To the best of our knowledge, there is no public dataset that contains the patches and related discussion data of Linux kernel development. To reduce effort in collecting and processing those data, we provide a dataset of a nine-year history of patches and related discussion recorded by the LKML project on the Linux kernel patchwork.[2]

To obtain this dataset was a laborious task. First, the raw data were difficult to download. Our dataset contains 666,550 patches produced from December 2008 to December 2017. Downloading and parsing 6,666 pages were necessary to retrieve the patch identifiers. In addition, for each identifier, we had to download two pages in order to retrieve the completed data of a single patch. Downloading them was time-consuming. Second, constructing structured data from raw data was difficult. Many plain text were present in the raw data and their formats were not unified. For example, eight date formats were used in the raw data and we had to unify them into a UNIX timestamp style. Third, we cleaned the data for the common needs of data use. For example, unifying developers' identifiers is always a challenge in mining software repositories.

The dataset we developed contains multiple levels of data (following the method adopted by Zhu et al. [9]). The Level 0 data are raw data from which anyone can extract fields that he or she needs (and which may or may not be provided in Level 1). The Level 1 data represent a structured dataset stored in MySQL. People can easily use and avoid processing (dirty) raw data by using Level 1 data. Level 2 data represent the results of further processing Level 1 data. This processing addresses the common needs of data use and thus requires that we have a thorough understanding of the practices of LKML. We summarize developer practice of LKML in Section 2.2, including how developers change their name and email in the community, how maintainers commit a patch to the source tree, and how developers send a multi-version or multi-slice patch to LKML. Based on these practices, we recover some lost (but valuable) information concerning patchwork, including identifying authors, associating commits with patches, and recovering patch-set relationships. People can conduct their studies based on our Level 2 data or combine the practices we summarize with their own findings to generate data that can be included in this level or generate the same data but with a higher precision than ours has (e.g., they may improve the method of identifying authors who have multiple identities).

2 OVERVIEW OF DATA SOURCE

The Linux kernel patchwork is a website that records the communication of Linux kernel developers about patches conducted by email. A patch is often sent by email and the discussion that follows is typically included in an email reply. To make this discussion public, these emails should be sent to a public mailing list so that

*Corresponding author.
[1]pull request is a primary method of contribution in GitHub today

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196475

[2]https://patchwork.kernel.org/project/LKML/list/?state=*&archive=both&page=1

subscribers to that mailing list can receive the patch and related discussion. The Linux kernel mailing list (LKML) is the primary mailing list for kernel development. The patchwork website records the mailing lists used by the Linux kernel community, including LKML.

The LKML project on patchwork recorded more than 666,550 patches sent to LKML from December 2008 to December 2017. In the following sections, we introduce the operating mechanism of this patchwork to explain how an email that comes into the patchwork becomes a patch or discussion. We explain how developers use LKML and which features of the recorded data can reflect those practices. We use such practices to obtain data features commonly required by relevant studies and therefore produce Level 2 data.

2.1 Operating Mechanism of Patchwork

For LKML, the patchwork website must have a server that subscribes to LKML. When LKML receives an email, it pushes it to all subscribers of LKML. When the patchwork server receives a push, it determines whether this email contains a patch. If it does contain a patch, the patchwork creates a new patch transaction related to this email and saves it to the database. If the email does not contain a patch, the patchwork determines whether it is a reply to an existing patch transaction. If it is, the email becomes a new comment to this existing transaction; otherwise, the patchwork drops this email. Therefore, in our Level 0 data, an HTML file records a patch transaction that includes both the patch and comments to the patch. Also included is an mbox file that records the patch purely. In our Level 1 data, the table "patch" records those emails that contain a patch, whereas the table "patch_comment" records the emails that do not contain a patch but that reply to another email that exists in the other table.

2.2 Practices of the LKML

The Linux kernel community has their own rules for using mailing lists. These rules can be reflected in the data and may affect later analysis in some way. We address the rules that involve common needs of related research and conduct further processing to obtain more clean (and complete) data. We also record how (and why) we process the detailed practices so that others can better understand the context of the data and use our dataset more efficiently.

Some developers may use different names and emails to send their patches to LKML. Usually, they do not simultaneously change their name and email; that is, they may change one or the other but not both.

The Linux kernel community requires developers to produce a patch that is small and simple [5]. Therefore, if a developer wants to make a complex contribution, he or she must split the patch into a series of small patches. This developer then sends a main email message to describe what he or she has done. After this main email has received responses, the developer sends a series of patch emails as additional replies in order to submit the patches and describe what each does. Because the main email does not contain a patch or reply to another patch, it is dropped by the patchwork. Therefore, we often see many patches with different IDs in their titles (e.g., "[05/17]," which indicates the fifth of 17 split patches). Those references to patches with IDs of "[01/17]" to "[17/17]" will be replies to an email that does not currently exist in the patchwork but exists in the LKML, and its title will contain "[00/17]," which means it is a main email.

Table 1: Basic Statistics of Dataset

#patch	#author	#patch linked to commits	#comment to patch	#patch set
666,550	9,243	139,664	1,208,320	136,139

Many titles of patches contain references such as "v2" or "v3" (obviously indicating the second or third version of a patch). Many patches need to be modified from their early version for various reasons (e.g., the implementation is insufficient or the descriptions are unclear). When the author submits a new version, he or she will send a new email to the mailing list and mark the version number in the title. People who use our dataset should be aware of this practice. They may detect a series of similar patches and thus can use this practice to recover the version relationship between similar patches.

When a maintainer[3] decides to accept a patch, in most cases, he or she will use the command 'git am' to apply a patch to his or her own source tree. Therefore, the commit time will be that when the maintainer executes the aforementioned command. However, no indication is given in a patch about this recorded author time. Moreover, in our study, we discovered that many author times exactly match the sending time of the patch email. However, some exceptional cases exist. If a patch is submitted by a maintainer, he or she may use other operations to merge his or her commit to the public source tree instead of 'git am'. In this case, a true author time will be preserved instead of the email send time.

3 OVERVIEW OF MULTI-LEVEL DATASET

The basic statistics of our dataset are listed in Table 1. Note that *#patch linked to commits* is the number of patches that could be successfully associated with commits (see Section 3.3.2).

3.1 Level 0 Raw Data

In Level 0 data, each patch is stored in two files: HTML and mbox. The HTML file is downloaded from one URL[4] and the mbox file from a different one[5] . The "PATCHID" in the URL is an integer that acts as the unique identifier of a patch on the patchwork. The two types of files of each patch are stored in the directory named "PATCHID" and the whole dataset of Level 0 contains 666,550 directories.

The HTML file contains some meta information about the patch email such as the author, send time, and reply-to. In addition, it contains the comments on the patch. The first comment contains the patch email's body but excludes the patch. The other comments are emails from other developers that reply to the patch. The HTML file also contains the patch body. However, this file is difficult to extract for Level 1. Therefore, we also download the mbox file to obtain the patch body easily.

To download these two type of files, we must know all the PATCHIDs of the LKML project on the patchwork. Therefore, we download the index page of the patches first.[6] The parameters

[3] A maintainer in the Linux kernel community is the person who has the commit right to the source tree that he or she maintains. The maintainer's job is to select the patch deemed adequate to merge into the source tree, and then to take responsibility for it.
[4] https://patchwork.kernel.org/patch/PATCHID/
[5] https://patchwork.kernel.org/patch/PATCHID/mbox
[6] https://patchwork.kernel.org/project/LKML/list/?state=*&archive=both&page=1

Table 2: Schema of Table "patch"

Field	Description
patchwork_id	The unique identifier on patchwork website
author_email	The mail address that sent this mail, extracted from mbox file
author_name	The name of the mail sender, extracted from mbox file
in_reply_to	The mail header named "In-Reply-To," extracted from HTML file
mbox	The raw content of mbox file
message_id	The mail header named "Message-Id," extracted from mbox file
reference_list	The mail header named "References," extracted from HTML file
send_time	The UNIX timestamp style time of sending this email, parsed from send_time_raw
send_time_raw	The mail header named "Date," extracted from HTML file
subject	The mail header named "Subject," extracted from mbox file

"state=*" and "archive=both" ensure that all patches will be listed. In addition, the parameter "page" indicates the page to download. When we conducted the downloading (December 10, 2017), there were 6,666 pages in the LKML project. Therefore, we downloaded index pages 1 to 6,666 and parsed them to obtain 666,550 PATCHIDs. Using those PATCHIDs, we then downloaded the two URLs described before for each patch.

3.2 Level-1 Structure Data

In this level, we extract all the important fields from the two files of Level 0 and store them in a MySQL database. We provide the SQL scripts used to recover the data from MySQL. Note that MySQL 5.7 is the proper version to execute such scripts, as the earlier versions of MySQL may produce errors when creating indices.

In the Level 1 data, we provide two tables of MySQL named "patch" and "patch_comment." Each line in Table "patch" represents a unique patch recorded by the patchwork (see Table 2). It contains a field labeled "patchwork_id," which represents the patch's unique identifier in the patchwork, and another field labeled "mbox," whose content is the same as that of the mbox file in Level 0. We also extract many other important fields such as author email, author name, send time, and email subject, and save them all in Table "patch."

The table "patch_comment" is extracted from the HTML files contained in Level 0 data. We extract the "comments" portion of the HTML file using Jsoup, which is a Java HTML parser. For each patch, multiple comments may exist. Therefore, like the other table, this table also contains a field labeled "patchwork_id" to indicate from which patch the comment derives, and another field labeled "id" to distinguish the comments that come from the same patch and to record the sequence. From the dataset, we extract 1,208,320 total comments.

3.3 Level-2 Further Processing Result

We use the Level 1 data combined with the Linux kernel source tree's git log as well as the practices of LKML (which we consider important for the common use of data) to generate additional results. These results represent an implementation of our observation regarding practices of LKML and we supplement them with the Level 1 data.

The processing results can be widely used in research related to patch. For example, in the studies on how to write a quality patch [6, 8], the researchers required data about whether a patch was accepted. Therefore, they had to associate the patch with the corresponding commit.

In our dataset, the processing results are provided as SQL scripts. To recover these data in MySQL, we suggest that people use MySQL 5.7 to execute those scripts.

3.3.1 Identifying Multiple Identities.

This processing step aims to identify the same patch author but who uses different names and emails. This step ensures that each patch is associated with a unique author.

We traverse all patches in order of email send time from earliest to latest. For each patch, we try to match the author email with an existing author. If the email matches, we associate the patch with this author; otherwise, we try to match the author name with an existing author. If the name matches, we associate the patch with this author; otherwise, we create a new author and associate the patch with this new author. Finally, we add the author's email and name to both the email set and name set of the matched author.

By using this method, we identified 9,243 authors. The results are presented in the tables "my_author," "my_author_author_email," "my_author_author_name," and "patch_author." The table "my_author" records all author IDs that we generate for those authors. The tables "my_author_author_email" and "my_author_author_name" record the authors' email and name sets. The table "patch_author" records the relationship between patch and author.

3.3.2 Associating Commits with Patches.

This processing step aims to connect a patch to a commit. Based on the results of this processing step, we know whether a patch has been accepted as well as the commit that has been produced as a result of this accepted patch.

First, we use the command "git log –format"[7] to dump the commit log from the Linux kernel mainline repository maintained by Linus Torvalds. If a commit appears in this repository, it will appear in the next version of the Linux kernel. After we obtain the commit log in plain text, we save it to the MySQL table "commit" for our next processing step.

As explained in Section 2.2, many commits exist whose author time is the same as the send time of the corresponding email. Therefore, our method dictates that for each commit, we try to find a patch whose send time is equal to the author time of the commit. If we identify none or multiple patches that match, we then try to find a patch whose subject contains the commit's subject.

Our results show that we successfully matched 139,664 patches. Specifically, 121,518 patches were matched by author time and the remaining 18,146 patches were matched by subject. These results are stored in the table "commit_to_patch."

3.3.3 Recovering Patch-set Relationship.

As explained in Section 2.2, a complex contribution may be split into a series of patches, and the relationship between these patches

[7] git log –format="%H%n%an%n%ae%n%aD%n%cn%n%ce%n%cD%n%s%n===END===" –no-merges > ../logdump

may get lost in practice. We call those patches "patch sets," and each patch in a patch set is called a "sub-patch." This processing step aims to recover the lost relationships between the sub-patches that belong to the same patch set.

We discovered that every sub-patch replied to a main email, but that the main email was not recorded by the patchwork. However, in the mail header, we found that a patch set's sub-patches had the same "InReplyTo" header, which means they replied to the same email. Therefore, we grouped the patches by the "in_reply_to" field, which was extracted from the mail header to recover the patch-set relationship.

The results are stored in the tables "patch_set" and "patch_set-_patchwork_ids." Overall, we found 498,856 sub-patches, and the number of patches whose subject contained "[*/*]" was 518,620.

4 USAGE

Substantial studies analyze patch data produced in open source projects to understand how the software is developed, what problems might be in the development process, and come up with methods or tools to help the development.

Baysal et al. [1] studied the lifecycle of patches in the Mozilla Firefox to understand the development process. To measure the efficiency of a community, Bird et al. [2] proposed a method to decide whether a patch was fully accepted or partially accepted or rejected. Jiang et al. [4] and Weißgerber et al. [8] studied the effort required for submitting and reviewing the patches to help the community to improve the efficiency of development. Zhu et al. [10] compared the effectiveness of patch-based tools and pull-request-based tools and found that pull request systems to be associated with reduced review times and larger numbers of contributions. To help developers write a good patch and reduce the effort of developers and reviewers, Tao et al. [6] and Weißgerber et al. [8] summarized the reasons for reviewers to reject a patch and located the features that make a patch accepted easily. Some studies proposed classification or recommendation methods for patch-based community. For example, Tian [7] proposed a method to identify the bug fix patch in the Linux kernel. Ibrahim et al. [3] proposed a method to recommend important discussions in mailing list to developers.

Our dataset can serve those researches well. With the level-0 and level-1 data, people can easily obtain and process a complete dataset with 9-year history of patches in LKML which contains the information of developers, details of patches and the discussions of patches. With the level-2 data, researchers can reduce their effort on processing the data for common needs. Knowing the practices which we summarized in Section 2.2, researchers can make better understanding of this dataset. For example, for studies about how to write a good patch, this dataset provides email body and patch body for feature extraction, provides comments (for each patch) which can be used to learn why this patch is good or bad, and it can tell that whether the patch is accepted or not in the further processing.

5 LIMITATION AND CHALLENGE

The LKML does not contain all patches sent to the Linux community. Thus, some commits exist that cannot be associated with a patch. Studies that require identifying these kinds of relationships must be aware of this limitation. Recognizing the multiple identities of a single author and recovering relationships between sub-patches in a patch set can never be accomplished perfectly because of the complexity of human (i.e., developer) behavior. However, in this study, we experimented with the best possible method to recover the relationships. For example, we found that a commit's author time can be used to match commits and patches. We also found that sub-patches replied to the same mail. Thus, our method can be used to recover relationships between sub-patches.

In general, our tools and methods can be used to extract and process (the same or other) mailing lists of the Linux community in the patchwork. Downloading and processing the raw data are not easy, particularly when processing the mbox file, which is a plain-text file. (The date and mail header in an mbox file have various formats and are thus difficult to process.)

6 SUMMARY

In this study, we presented a multi-level dataset of the LKML project on the Linux kernel patchwork. This dataset can reduce researcher effort in collecting, cleaning, and processing patch data. We also presented the methods and tools to generate this dataset. These can be used to extract the latest data or the data of similar projects. Furthermore, our observations about the practices of the LKML (e.g., how developers send a multi-version or multi-slice patch to the LKML) may help people understand the context of the data being produced and thus help them better utilize our dataset.

7 ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China Grants 61432001 and 61690200, and the National Basic Research Program of China Grant 2015CB352200.

REFERENCES

[1] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. 2012. The secret life of patches: A firefox case study. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 447–455.

[2] Christian Bird, Alex Gourley, and Prem Devanbu. 2007. Detecting patch submission and acceptance in oss projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 26.

[3] Walid M Ibrahim, Nicolas Bettenburg, Emad Shihab, Bram Adams, and Ahmed E Hassan. 2010. Should I contribute to this discussion?. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 181–190.

[4] Yujuan Jiang, Bram Adams, and Daniel M German. 2013. Will my patch make it? and how fast?: Case study on the linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 101–110.

[5] Andi Kleen. 2008. On submitting kernel patches. In *Linux Symposium*. 253.

[6] Yida Tao, Donggyun Han, and Sunghun Kim. 2014. Writing acceptable patches: An empirical study of open source project patches. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 271–280.

[7] Yuan Tian, Julia Lawall, and David Lo. 2012. Identifying linux bug fixing patches. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 386–396.

[8] Peter Weißgerber, Daniel Neu, and Stephan Diehl. 2008. Small patches get in!. In *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 67–76.

[9] Jiaxin Zhu, Minghui Zhou, and Hong Mei. 2016. Multi-extract and Multi-level Dataset of Mozilla Issue Tracking History. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 472–475. https://doi.org/10.1145/2901739.2903502

[10] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. 2016. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 871–882.