

GIT-REVIEWED: A DISTRIBUTED PEER REVIEW  
TOOL & USER STUDY

MURTUZA I. MUKADAM

A THESIS  
IN  
THE DEPARTMENT  
OF  
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE  
CONCORDIA UNIVERSITY  
MONTRÉAL, QUÉBEC, CANADA

APRIL 2014

© MURTUZA I. MUKADAM, 2014

CONCORDIA UNIVERSITY  
School of Graduate Studies

This is to certify that the thesis prepared

By: **Murtuza I. Mukadam**

Entitled: **git-reviewed: A Distributed Peer Review Tool & User  
Study**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Computer Science**

complies with the regulations of this University and meets the accepted standards  
with respect to originality and quality.

Signed by the final examining committee:

\_\_\_\_\_ Chair  
Dr. H. Harutyunyan

\_\_\_\_\_ Examiner  
Dr. N. Tsantalos

\_\_\_\_\_ Examiner  
Dr. R. Witte

\_\_\_\_\_ Supervisor  
Dr. P. Rigby

Approved \_\_\_\_\_  
Chair of Department or Graduate Program Director

\_\_\_\_\_ 20 \_\_\_\_\_  
Christopher Trueman, Ph.D., MEng, BEng, Interim Dean  
Faculty of Engineering and Computer Science

# Abstract

## git-reviewed: A Distributed Peer Review Tool & User Study

Murtuza I. Mukadam

Software peer review has been considered the basis of an effective procedure for examining software artifacts, identifying defects and increasing the efficiency of software firms for decades. The process of peer reviewing allows reviewers to check their co-worker's work, which helps in determining if a standard for a system has been maintained or achieved by the person whose work is being reviewed. The result of this process is a high quality working product that will likely reduce further maintenance effort. There is a large number of code review tools available in the market as of today, which assist in the reviewing task. All these tools are centralized, with the reviews and discussions being stored either on a mailing list or a server. In contrast, the code that makes up a software system is increasingly being stored in a distributed version control system (e.g. Git). In an effort to determine if distributed peer review is a tenable idea, we develop a peer review tool, git-reviewed, which replicates the working model of Git by incorporating reviews into the current Git architecture. git-reviewed is a lightweight and a truly distributed peer review tool, which eliminates a centralized server or a mailing list to store the review discussions. We model our use case based on the Linux kernel, which is a large open source project. git-reviewed has been designed to keep the current development and reviewing practices followed by the Linux kernel developers within the open source environment intact. We also provide better traceability by linking the review discussions on the mailing lists and the changes made in the Git repository. git-reviewed was evaluated by software developers working on large open source projects, (e.g. Linux, PostgreSQL, Git), whose feedback helped us improve our tool and determine if distributed reviewing works in practice.

# Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Peter C. Rigby who has been actively involved in mentoring me and keeping me well aligned with my goals, throughout the course of this thesis. His consistent guidance and motivation has been instrumental in bringing the best out in me. He has not only taught me a lot about the best research practices, but has also inspired me to achieve great heights in life. This thesis would not have been possible without Dr. Rigby's expertise and sound research insights in the field of Software Engineering.

I would also like to thank Dr. Daniel German for collaborating with me and providing me with the much needed assistance for this work.

A special mention to Tavish Armstrong, a friend and a colleague. I am particularly indebted to him for his selfless help and some crucial contributions to this work.

I am extremely thankful to all the members of the Concordia Empirical Software Engineering Laboratory (CESEL) for making my work environment so lively. Research would have never been so enjoyable without them.

Last but not the least, I wish to express my love to my parents, brother and my lovely fiancée. No words can justify their importance in my life. I would have never made it so far without their constant support. I owe every smallest achievement to them.

# Contents

List of Figures	viii
List of Tables	ix
<b>1 Introduction</b>	<b>1</b>
<b>2 Context and Motivation</b>	<b>4</b>
2.1 Relevant Linux Kernel Background . . . . .	4
2.2 Peer Reviewing on Linux Kernel Mailing List . . . . .	5
2.3 Need for Better Traceability . . . . .	9
2.4 Need for Distributed Review . . . . .	10
2.5 Advantages and Disadvantages of Distributed Development Tools . . . . .	10
2.5.1 Elimination of a Central Point of Failure . . . . .	11
2.5.2 Collaboration . . . . .	11
2.5.3 Commit Policies . . . . .	12
2.5.4 Release Engineering . . . . .	12
2.5.5 Trusting Your Data . . . . .	13
<b>3 Background</b>	<b>14</b>
3.1 Evolution of Peer Reviewing Practices . . . . .	14
3.2 Summary of Existing Tools . . . . .	17
3.2.1 CodeCollaborator . . . . .	18
3.2.2 Crucible . . . . .	19
3.2.3 Review Board . . . . .	19
3.2.4 Rietveld . . . . .	19
3.2.5 Gerrit . . . . .	20

3.2.6	Bugzilla Based Review . . . . .	20
3.3	GitHub Pull Requests . . . . .	21
3.4	Email Based Code Review . . . . .	21
<b>4</b>	<b>Architecture, Implementation, and Features of git-reviewed</b>	<b>23</b>
4.1	High-level Features for git-reviewed . . . . .	24
4.2	Git Architecture . . . . .	25
4.2.1	Branching . . . . .	27
4.2.2	git-reviewed Integration with Git . . . . .	28
4.3	Initial Architecture of git-reviewed . . . . .	28
4.3.1	Initial Architecture of git-reviewed . . . . .	28
4.3.2	Current Architecture of git-reviewed . . . . .	30
4.4	Implementation of git-reviewed . . . . .	33
4.4.1	git-reviewed Option Parser . . . . .	34
4.4.2	Communication with Remote Repository . . . . .	34
4.4.3	Creating and Deleting Reviews . . . . .	34
4.4.4	Creating and Sending Emails from git-reviewed . . . . .	34
4.4.5	Developing Log History of Reviews . . . . .	35
4.5	Features and Functionality . . . . .	35
4.5.1	Viewing a List of all Reviews . . . . .	35
4.5.2	Log of Reviews . . . . .	37
4.5.3	Viewing Each Review . . . . .	37
4.5.4	Viewing Reviews Related to a Particular Reviewer . . . . .	37
4.5.5	Creating a Review . . . . .	37
4.5.6	Importing Reviews from a Mailing List . . . . .	37
4.5.7	GitHub Compatibility . . . . .	42
4.6	Need of Linux Reviewers and Features of git-reviewed . . . . .	44
4.7	Linking Patches to Commits . . . . .	46
4.7.1	Comparing Linking Results for Linux, Git and PostgreSQL . . . . .	49
4.8	Comparing Linux Reviewing Process and git-reviewed Reviewing Process	50
<b>5</b>	<b>Evaluation</b>	<b>52</b>
5.1	Distributed Review . . . . .	53

5.2	Traceability . . . . .	53
5.2.1	git-reviewed Tracker . . . . .	54
5.2.2	Comparison with Trackgit . . . . .	55
<b>6</b>	<b>Conclusion and Future Work</b>	<b>57</b>
6.1	Conclusion . . . . .	57
6.2	Future Work . . . . .	58
	<b>Bibliography</b>	<b>59</b>
<b>A</b>	<b>Email Invites</b>	<b>64</b>
A.1	Recruitment Email . . . . .	64
A.2	Discussion with Takashi Iwai . . . . .	65
<b>B</b>	<b>git-reviewed User Manual</b>	<b>66</b>
B.1	Installation . . . . .	66
B.2	Man Page . . . . .	66

# List of Figures

1	LKML message-Id 20140113235650.GA23380@kroah.com . . . . .	7
2	Chain of trust hierarchy in Linux kernel development process . . . . .	8
3	Git Architecture, modified from [34] . . . . .	26
4	Initial architecture of git-reviewed . . . . .	29
5	Invalid object error in Git . . . . .	30
6	Architectural diagram of git-reviewed . . . . .	31
7	An instance of git-reviewed . . . . .	32
8	git-reviewed displaying the list of reviews . . . . .	36
9	git-reviewed displaying the log of reviews . . . . .	38
10	Viewing each review on git-reviewed . . . . .	39
11	Viewing reviews for a particular reviewer on git-reviewed . . . . .	40
12	Creating a review on git-reviewed . . . . .	41
13	Commits reviewed on GitHub . . . . .	43
14	Reviews for a specific commit on GitHub . . . . .	44
15	Review on GitHub . . . . .	45
16	Uploading a review to the mailing list git-reviewed . . . . .	48
17	Track reviews online . . . . .	55



# List of Tables

1	Comparison of modern code review tools . . . . .	18
2	Git commands compared to git-reviewed commands . . . . .	47
3	Comparing Linux reviewing process with git-reviewed reviewing process	51

# Chapter 1

## Introduction

For over three decades, software engineers have worked tirelessly to develop methods and tools to improve the quality of software artifacts and ensure that a bug free software product is delivered in the least possible time. Over the years, a number of software firms have adopted code review tools depending on the requirements and the features these tools have to offer. A common theme amongst all the code review systems present today is that all of them are centralized. This means that the reviews and discussions are stored on a central server (e.g. Gerrit) or a mailing list (e.g. Linux Kernel Mailing List).

Moreover, in recent years, many software projects are stored in distributed version control systems (e.g. Git), which allow users to fully mirror the repository on to their local machines. This also enables users to be independent of a centralized server and to decide which code they will permit into their own repositories. Further, even if the server hosting a Git repository crashes, any of the developers can copy back their local repository to the server and restore it [6]. Torvalds, the chief architect of the Linux kernel, explains the advantages of working in a distributed environment [45]. From backing up projects to commit access issues, distributed development proves to be an ideal way to work on an open-source software (OSS).

The reviewing practices that have been adopted by the Linux kernel developers have worked well for over 20 years. However, there is lack of traceability and there is need to provide better and simpler ways to link review discussions on the mailing lists to the respective commits in the Git repository [41].

Additionally, Shawn Pearce, the creator for Gerrit code review tool mentioned the

advantages of having a distributed code review. Distributed review would allow an individual to decide and store only the reviews which are interesting and to make reviews in an offline manner. In not making Gerrit distributed, Pearce regrets the lost opportunity to combine a distributed version control system Git, with a distributed reviewing system [43].

This thesis has two research goals. First, is distributed peer review tenable in practice? Second, can we add better traceability to the current Linux reviewing practices? In order to determine this, we developed and implemented a distributed peer review tool, git-reviewed, that minimally modifies Git and incorporates reviews seamlessly into the current Git architecture. We model our use cases on the Linux kernel in order to encourage the adoption of our tool. git-reviewed is distributed with no central review server. This means that users can review patches without visiting any website or subscribing to a mailing list. Like the source code and its history, reviews are stored locally, so developers can perform them without an internet connection and keep only those reviews that are of interest to them. We also add traceability to the current Linux kernel reviewing practices.

We evaluate our tool by sending out email invites to the developers working on different projects which use Git as their version control system. Their suggestions and feedback was taken into account which led to improvements in our tool and achieve our research goals.

Our research follows the following steps and this thesis is divided into different chapters:

**Analysis of peer reviewing processes:** We analysed the tool supported reviewing practices adopted by developers. Two motivations emerged from this analysis: better traceability in the current Linux kernel reviewing practices and allow reviewers to perform distributed reviews. The need for distributed review and better traceability in the current Linux kernel reviewing practices is described in detail in Chapter 2. We also describe the features that must be present in git-reviewed to make it amenable to the Linux and other OSS developers.

**Survey of modern peer review tools:** We conducted a survey of how modern peer review tools allow developers to perform reviews. We found that none of the tools currently present in the market support distributed review. A study on how

peer reviewing practices evolved and a comparison of the current peer review tools helped us in realizing some disadvantages of having centralized peer review tools, and so, the purpose of having a distributed peer review tool became even more justified. Chapter 3 provides the differences between the modern peer review tools and an overview of the evolution of peer reviewing practices.

**Develop and implement a distributed peer review tool:** Subsequently, we designed and developed a distributed peer review tool, which minimally modified the Git architecture. We were able to incorporate review into Git, which would allow developers to perform reviews in an offline manner as well. Chapter 4 looks at the design decisions made while building git-reviewed. We also demonstrate that the git-reviewed tool functions in the same way as reviews are performed on the Linux kernel mailing list.

**Evaluation:** We contacted developers working on Linux, Git and PostgreSQL projects in order to gather feedback regarding the use of our tool. We use an interview methodology to gather feedback. Our goal was to address our research question from the developer's point of view. In Chapter 5, we provide the results of our evaluation. We ascertain the effect of git-reviewed on the Linux community and other OSS projects and determine if distributed review is a good fit into the current Linux reviewing practices. our tool.

**Modifications based on feedback:** The feedback we received from the developers helped us make modifications and provided with the much needed answers to our research question. We conclude this thesis in Chapter 6 and discuss possible improvements and additional features that can be added to git-reviewed.

## Chapter 2

# Context and Motivation

The goal of this work is to design, develop, and implement a truly distributed peer review tool in order to determine if distributed peer review is a tenable idea. Since the most widely used distributed version control system, Git [22], was designed for and created by the Linux kernel community, we decided to target our git-reviewed tool to the same audience. In this chapter, we give a background on the Linux kernel, describe the advances to version control systems that were introduced by Git, and provide the motivations behind developing git-reviewed which would also add traceability to the current Linux kernel reviewing practices.

## 2.1 Relevant Linux Kernel Background

The Linux kernel project was initiated by Linus Torvalds in 1991 as a part of his personal project while studying at the University of Helsinki in Finland. It was supposed to be an alternative to the already present operating systems like Windows, Mac OS, MS-DOS and others. Torvalds' main motive behind building Linux was to deliver an operating system to the users which would be open to comments and suggestions, unlike the version of Unix operating system called 'Minix', whose creators would not entertain any requests for improvements [42]. Linux although built on the basic concepts of Unix, it is highly user-friendly and easier to install as compared to Unix.

The Linux kernel is one of the largest open source projects. It is highly efficient and robust. It has over 8 million lines of code and thousands of contributors working

for over 100 different companies contributing to the Linux kernel project.

The Linux source code can be modified or distributed under the GNU General Public License. This means that there are many Linux distributions available in the market, most of them include the Linux kernel and have supporting utilities built on top of it.

There is a large number of versions and forks of the Linux project. However, the ‘mainline kernel’, which is maintained by Linus Torvalds is used as the basis and trusted by the many Linux distributors. There are tremendous advantages of contributing and working with the mainline kernel as it is available to all Linux users. It is often improved over time and is constantly reviewed by various developers worldwide.

Almost all of the reviewing that takes place happens on electronic mailing lists. An asynchronous reviewing process is followed. There are many such mailing lists where discussions, announcements and reviewing take place [8].

Until 2002, the Linux kernel project was maintained and managed solely with the help of patches and tarballs. Any changes to the system was done with the help of files. There was no version control system to maintain Linux. In 2002, a proprietary distributed version control system, BitKeeper, took over the task of maintaining of Linux [6].

Git, a distributed version control system was developed in 2005 after the proprietors of BitKeeper and the community which developed Linux kernel parted ways after a disagreement. Torvalds and a few other contributors began creating Git in April 2005. Git was particularly targeted to withstand the corruption of files. It was also designed in a way to handle extremely large projects like the Linux kernel itself [45][46].

git-reviewed is built on top of Git, extensively using the Git API in order to make the best use of the advantages the distributed version control system has to provide.

## 2.2 Peer Reviewing on Linux Kernel Mailing List

Asynchronous, electronic code review is a natural way for OSS developers, who rarely meet in person, to ensure that the community agrees on what constitutes a good code contribution. Most large, successful OSS projects see code review as one of their most important quality assurance practices [40, 31, 17]. Rigby [41] describes the reviewing

process which takes place on OSS projects. A patch, which is a set of changes are created by a developer if he is interested in making significant contributions to the project. The following steps explain the reviewing process:

- The author submits a contribution by emailing it to the developer mailing list or posting to the bug or review tracking system.
- One or more people review the contribution.
- It is modified until it reaches the standards of the community.
- It is committed to the code base. Many contributions are ignored or rejected and never make it into the code base [21].

Similarly, the Linux kernel developers have adopted a highly asynchronous email based reviewing practice. There is a large number of mailing lists, where developers upload changes and review discussions take place. The most fundamental communication channel is the Linux Kernel Mailing List (LKML) which is the hub for all the major announcements and discussions for the Linux kernel. The Linux kernel mailing list is an extremely busy mailing list with over 400 messages being sent on it per day [9].

The example in Figure 1 illustrates a review on the LKML. <sup>1</sup>

Since each message on the mailing list is given a unique message-id, it is easy to distinguish it from the other. The diff is uploaded and there are comments made by the reviewer (Greg Kroah-Hartman).

We try to implement a tool, which will not change the working practices of the Linux developers, thereby, making it easy for them to adopt the tool. We study the needs of the Linux reviewers and try to make use of them in our tool. The features implemented in git-reviewed are described in Chapter 4 followed by their comparison to the needs of the Linux developers.

```

Date      Mon, 13 Jan 2014 15:56:50 -0800
From      Greg KH <>
Subject   Re: [PATCH] Staging: ced1401: fix coding style in ced_ioc.c

On Mon, Jan 13, 2014 at 08:41:40PM +0100, Pol Eyschen wrote:
> From: Pol Eyschen <poleyschen@hotmail.com>
>
> All comments fixed to match the kernel coding style.
>
> Signed-off-by: Pol Eyschen <poleyschen@gmail.com>
> ---
> drivers/staging/ced1401/ced_ioc.c | 382 ++++++-----
> 1 file changed, 249 insertions(+), 133 deletions(-)
>
> diff --git a/drivers/staging/ced1401/ced_ioc.c b/drivers/staging/ced1401/ced_ioc.c
> index 62efd74..e5172cb 100644
> --- a/drivers/staging/ced1401/ced_ioc.c
> +++ b/drivers/staging/ced1401/ced_ioc.c
> @@ -41,7 +41,8 @@ static void FlushOutBuff(DEVICE_EXTENSION *pdx)
> {
>     dev_dbg(&pdx->interface->dev, "%s currentState=%d", __func__,
>         pdx->sCurrentState);
> -    if (pdx->sCurrentState == U14ERR_TIME) /* Do nothing if hardware in trouble */
> +    if (pdx->sCurrentState == U14ERR_TIME)
> +        /* Do nothing if hardware in trouble */
>     return;
>
Please wrap stuff like this in { } to not confuse people.
thanks,

greg k-h

```

Figure 1: LKML message-Id 20140113235650.GA23380@kroah.com



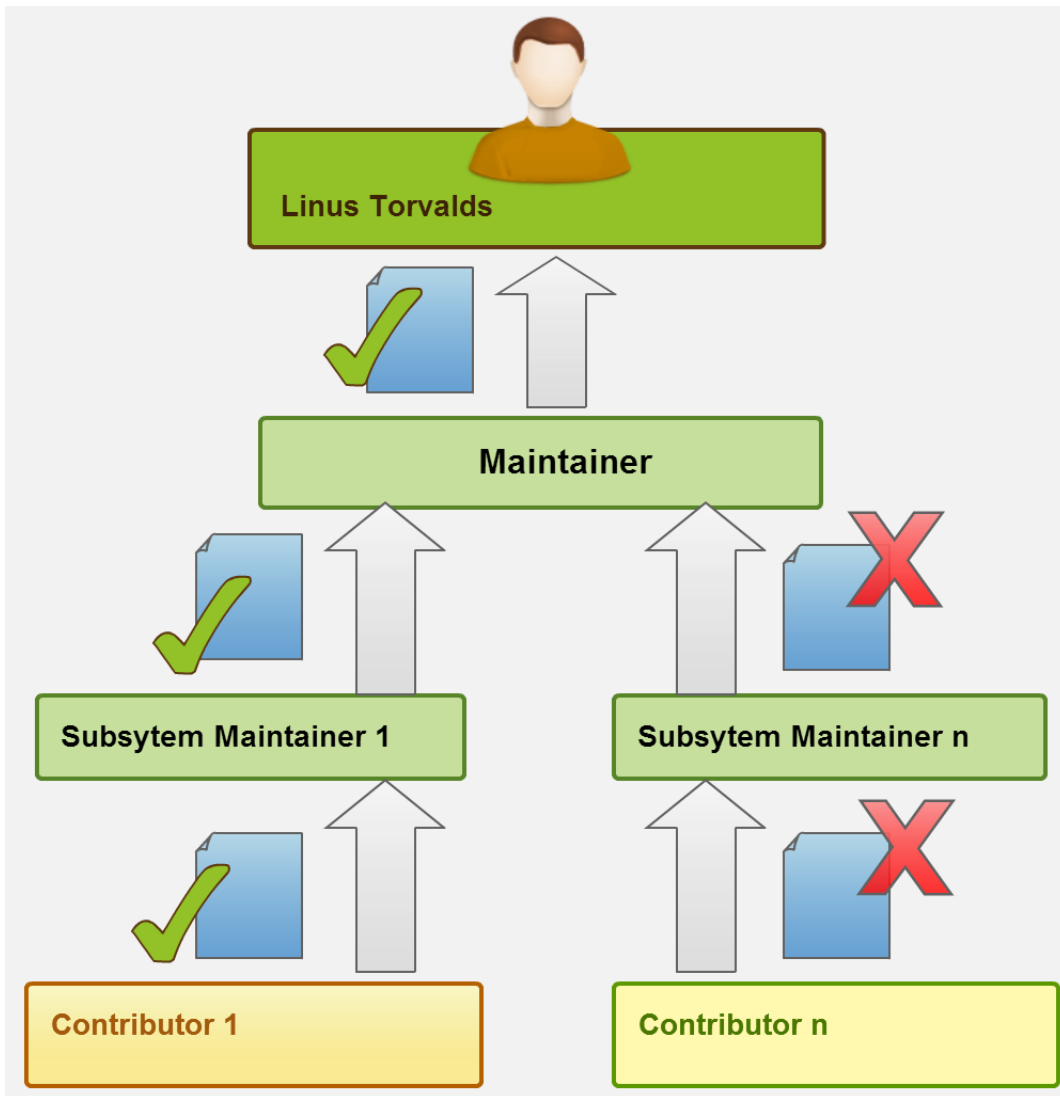


Figure 2: Chain of trust hierarchy in Linux kernel development process

## 2.3 Need for Better Traceability

Figure 2 shows the chain of trust hierarchical model in the Linux development. There are thousands of contributors uploading patches on the Linux kernel mailing list everyday, which get reviewed over a period of time [9]. This means that a lot of care needs to be taken when any change is made to the system as the chances of bugs creeping in multiplies. Patches uploaded by any contributor do not directly get merged into the main system. It undergoes a rigorous process of reviewing in order to maintain the quality of the system. A developer first sends a request for reviewing to a kernel's subsystem mailing list in order to have other peer developers look at their work. Patchsets are posted to one of the relevant mailing lists. This allows the other developers to review the patch and give their feedback and comments. This first step towards the final merge is called the 'early review'. The developer needs to incorporate the feedback and re-submit the patch in order to get it through the reviewing stage. The developer has to convince the subsystem maintainer that the patch will make an impact without breaking the system. This process is known as the 'wider review'. After getting a positive response from all the reviewers especially the maintainer of the subsystem, the successful patch is finally merged into the main repository managed by Linus Torvalds which would appear in the next official Linux kernel release [8]. Process of reviewing lasts anywhere between a couple of days to years [9].

Since there are so many discussions taking place on the Linux kernel mailing list, and minimal or no information provided in each message regarding the commit or which part of the system is being discussed, it becomes extremely difficult to find which review is associated to which commit.

Our goal is to add traceability by providing ways to determine the review discussions revolving around a particular commit without using a search engine or searching extensively in the archives.

---

<sup>1</sup><https://lkml.org/lkml/2014/1/13/906>

## 2.4 Need for Distributed Review

The major difference between a distributed version control and a central repository is that in distributed version control, the developer is in complete control of what makes into the final repository. Rigby *et al.* [37] found out that a majority of developers preferred to work with local copies of the projects on their machines in order to trace the version history, make branches and perform commits, but eventually push them all on to a central repository which has the most updated code. In effect, it is not truly distributed. However, the Linux project has no central copy. There is however, only a large number of different versions of Linux apart from the ‘mainline kernel’ which has only one true committer: Linus Torvalds. The Linux project is authentically distributed with over 600 repositories owned by various developers [8].

We aim to include the various review discussions which are related to the changes made in these repositories along with the changes itself. We intend to distribute the reviews along with the contents of these repositories. In order to do so, we incorporate reviews into the current Git architecture and also make the best use of the advantages of distributed version control systems.

Pearce, the Gerrit code review tool’s primary designer, laments not making Gerrit distributed [43]. He admits that by making Gerrit a distributed tool, he would make it easier for the remote teams to collaborate with each other. Carrying reviews around would be just be like carrying the Git history anywhere.

Distributed review will also allow developers to perform reviews in an offline manner as all the reviews will be present locally. It would also provide the developer the freedom to store only reviews which are important to him.

Some of the other advantages of distributed development tools and how git-reviewed makes use of it is explained in the next section.

## 2.5 Advantages and Disadvantages of Distributed Development Tools

Torvalds [45], describes how it was very important according to him to make a version control system distributed. He briefly talks about the advantages and disadvantages

of working with a distributed version control. Below, we take this commentary and apply it to the design of git-reviewed.

### **2.5.1 Elimination of a Central Point of Failure**

The distributed version control systems eliminates the centralized server and so, the risk of a central point of failure is eliminated too. Everyone has their own copy of the data and so the risk of losing the history due to corruption or other failures is minimal. This is considered to be one of the most important features of a distributed version control system. If the server fails, restoration is easy as any user's local copy of the repository can be copied back to the server. Through compression and other techniques the storage is made efficient [34]. For example, the history of the entire Linux repository currently repacks itself into a 175MB pack for 63428 commits [33]. Like Git, git-reviewed allows the users to have all the reviews present locally on their machines and it also completely reduces the risk of losing the reviews due to a centralized server failure. One issue with distribution is that each developer has a duplicate copy of the entire history of the system. We assess the level of duplication in git-reviewed.

### **2.5.2 Collaboration**

A developer can either choose to work on the most recent commit (head) or on a release or tag. "Working off the Head", instead of using branches, allows developers to work with the latest code and avoid merging issues. But, it also heightens the possibilities of having an unstable and buggy code as the developer's attention could be diverted as they would be constantly switching between integration and development changes. Torvalds strongly suggested that the developers should work off stable tags or releases as developers can remain isolated from any unrelated changes.

Branches affecting the main repository can be a real cause of problems. Since Git allows users to carry the version control history anywhere and continue working and check all logs, commits, it also means that everybody who can commit effectively on their own would in a way have their own branch resulting in a lot more branching. When working on a centralized repository, merging a branch can be messy or time

consuming. And so, Git was designed to give its user the flexibility to have as many branches as required and hence, allowing the user to have some branches dedicated strictly to some experimental work. The user can work on a separate branch without worrying about the changes being made on the master branch.

git-reviewed has been designed keeping in mind this feature which Git provides. git-reviewed has its own 'review' branch where all the reviews are stored and it does not interfere with the master branch. Users can now work and make reviews without disturbing other co-reviewers. This also makes sure that the risk to the Git repository is minimal because git-reviewed only modifies the 'review' branch that it creates to store reviews related to the commits in the repository. git-reviewed does not modify Git and just runs standard Git commands. To remove git-reviewed, one simply has to remove the 'review' branch from the Git repository.

### **2.5.3 Commit Policies**

Determining who should be allowed to commit to a repository is eliminated with distribution. Each developer who has a copy of the system can choose which commits he or she will include. The problem of commit access can be eliminated. It thereby, results in the reduction of the politics involved in granting access. git-reviewed also allows the developers to save only the reviews which he or she is most interested in. A developer can maintain the reviews on the review branch and can delete all the reviews for any other reviewer by running one command. This can also help with any storage problems allow users to remove unrelated reviews thereby solving the duplication issue.

### **2.5.4 Release Engineering**

The process of release engineering is often described as the integration of various processes like tracking changes, automated tests and source code delivery to obtain a finished and efficient software artifact [20]. Distributed systems can help in a good release process. One can have a verification team, and they can pull in modifications from the developing team and verify it. After verification they pass on the modified product to a release team, thereby making the release process simpler and more

efficient.

git-reviewed provides traceability between commits and reviews. As a result, release engineers can check if each commit has been reviewed before release. The lack of traceability in email based review, means that checking if all commits have been reviewed is practically impossible (e.g. Linux has thousands of new changes in each release [8]).

### **2.5.5 Trusting Your Data**

With previous version control systems, corruption of a file on the system could go unnoticed. However, with Git, every file, commit, etc. has its own SHA-1 hash code. Git quickly detects corruption by comparing the content of a commit or file with the SHA-1 hash. Since each review has its own hash code, a similar check can be performed.

We will now move on to the evolution of peer reviewing practices over the years. We also compare the modern peer review tools on various parameters in the next chapter.

# Chapter 3

## Background

In this chapter, we will give a brief introduction of the evolution of code reviewing practices right from the time when Michael Fagan introduced software inspections. We discuss the various peer reviewing tools like CodeCollaborator, Crucible, Review Board, Rietveld, Bugzilla, Gerrit along with email based reviews and the GitHub pull requests to understand how software peer reviewing is conducted today. We closely examine these tools and techniques and note that all of them are centralized. In order to make a distributed peer review tool, we studied how tool supported reviewing is done on a centralized server along with its advantages and disadvantages and finally designed and implemented git-reviewed, which does not need a central server and reviewing can be performed in a distributed manner.

### 3.1 Evolution of Peer Reviewing Practices

Fagan introduced software inspections as a technique to increase the quality of software systems [26]. His idea of conducting ‘software inspections’ proved to be highly effective and adopted by many. This was the beginning of a legacy which till date has been useful in the development of high quality software. Fagan believed that examining software design and code in an organized manner not only played a critical role in reducing user reported defects, but it also increased the overall productivity drastically. “*The Fagan Inspection Process is as relevant and important today as it was thirty years ago, perhaps even more so*” [27].

Inspections used the following processes. The agenda was set in advance with the

participants actively trying to find bugs. The code along with the relevant material was circulated well before the meeting. The meeting would include an author, a number of software engineers who would study the code, a meeting chair and a person who recorded the discussion. Fagan's inspection process was characterized by an entry and exit criteria where the output of each operation was compared to the exit criteria in order to keep track of the number of bugs found in the process. Fagan was among the first ones to provide empirical evidence that it was cheaper to fix the bug as early as possible than to wait for software testing or customer reported failures [26].

Brothers *et al.* modernized Fagan's ideas to the 1990's by creating a software system called the ICICLE (Intelligent Code Inspection Environment in a C Language Environment). This tool computerized the inspection process by eliminating paper and easing the methods of reporting defects [23]. Its target was to solve the issues faced during the process of code inspection which included the inability of a single developer to understand the code written by another developer in the given time constraints and the poor efficiency of some code inspection meetings. ICICLE provided an efficient solution to the code inspectors by creating a computer aided environment to check basic programming errors, standard code violations and browse through manual pages and library function specifications. The human interface was the most critical component of this software system. ICICLE also reduced the effort put in by the team of inspectors discussing a problem by providing support for paperless communication between them during a code inspection meeting. Brothers' solution to some of the flaws in the old technique of manual inspections resulted in an overall increase in the number of inspections in lesser time in spite of keeping the roles of the participants and the basic structure of the meetings intact [29].

Although Fagan's idea of inspection process was repeatedly shown to be effective at finding defects [27], its efficiency was questioned. Votta [47] argued the need of having inspection processes. An analysis of inspection meetings that took place from May 1991 to December 1991, showed that the cost of these meetings was actually higher than what it was thought to be. It was found that 20% of the inspection interval was related to scheduling conflicts [30]. Thus, "*Does every inspection need a meeting?*", became the most arguable query in the history of code reviewing [24]. Votta [47] provided several alternatives to the manual inspections. He suggested



having a three person meeting which included an author, a moderator and a reviewer to gather the findings and comments. He also suggested that exchange of ideas or information between an author and reviewer could also be possible without them actually meeting. Verbal, handwritten or electronic forms of communication were advised. Some problems associated with the inspection process more often than not overshadowed the effectiveness of the formal meeting based reviewing process. There were cases of reviewers not preparing themselves enough before the meeting. Not all problems could be discussed in a meeting due to the time constraints. Perry studied that asynchronous meetings found as many defects as co-located meetings [32]. These various factors paved way for the asynchronous type of reviewing.

Wiegers [48] studied the effects of asynchronous reviews. It was believed to target at the deficiencies of conventional peer reviews. Asynchronous meetings disallowed the participants from detouring on different and vague topics while in a meeting. It also broke the geographic barriers where reviewers could participate according to their own convenience. Asynchronous reviews worked best for the distributed teams.

Rigby *et al.* [39] studied the differences between the OSS peer reviewing process to the traditional inspection process and provided solutions which could make the development process of proprietary software efficient. They suggested the usage of lightweight peer reviewing tools which would provide more traceability and track changes. The overly formal process was replaced by lightweight review that relied on experts.

Peer reviewing was not just restricted to finding defects. Bachhelli *et al.* [18] performed a study on diverse teams at Microsoft and realized that software code review also allows knowledge transfer and team awareness as reviewers could share their different viewpoints with other reviewers and provide different solutions to a problem.

The evolution of the process of code reviewing in the past 30 years has been remarkable. It has been pivotal in the success of many open source projects.

We will discuss the working of some of the major peer code review tools in the rest of this chapter.

## 3.2 Summary of Existing Tools

The formal process introduced by Fagan [26] has been replaced by practitioners with a lightweight tool supported review process [38]. These lightweight tools continued to evolve and quickly became the hub for collecting the files, reviewing code, and finding defects. In this section, we examine the current peer review tools that are used in industry.

We contrast the tools based on the following (Table 1 provides a summary of this comparison):

- **Branching:** This feature allows the developers to upload patch sets, files on a mailing list or server in order to be reviewed by the other co-developers.
- **Traceability:** We compare the peer reviewing tools based on how well the tool offers linking between the review discussions and the commits or the changes made in a repository. This is an important feature which is taken into account by most peer reviewing tool developers as with the increasing number of commits and discussions, it can become extremely cumbersome to find discussions revolving around a problematic commit.
- **Integration with Version Control:** Most of the tools are integrated with version control systems (e.g. Git, SVN, Mercurial). The users of peer review tools prefer the version control systems to handle various processes like merging, committing [43]. This integration also reduces the adoption overhead as majority of the software system source code is already hosted on the version control systems. Almost all the peer review tools allow integration with more than one version control systems.
- **Web Interface:** A web interface with these tools allow better navigation through the review discussions over the Internet. This allows the tools to achieve a greater level of interoperability with isolated desktop systems.
- **Cost:** While most of the tools are freely available, there are high costs associated with some of these tools.
- **Distributed:** This is the most important feature targeted in this thesis. While

all the tools are bound to a centralized server, git-reviewed is the only truly distributed peer reviewing tool.

Table 1 compares the above discussed code review tools on the basis of different features and parameters.

Tools	Branching	Traceability	Integration with V.C	Web Interface	Cost	Distributed
Email	Patchsets	Low	No	No	Free	No
Gerrit	Patches	High	Yes	Yes	Free	No
GitHub	Pull Requests	High	Yes	Yes	Free	No
Rietveld	Patches	High	Yes	Yes	Free	No
Code Collaborator	Files, Change Lists	High	Yes	Yes	High	No
Crucible	Files	High	Yes	Yes	High	No
Review Board	Patches	High	Yes	Yes	Free	No
git-reviewed	Review Branch and Patches 4.3	High 4.7	Yes 4.2.2	Yes 5.2.1	Free	Yes 4.1

Table 1: Comparison of modern code review tools

### 3.2.1 CodeCollaborator

Cohen [24] explains how reviews are conducted at Cisco using the CodeCollaborator’s web based user interface. Authors decide who will participate in the review and accordingly email invites are sent to the potential participants using the tool. Ratcliffe [35] describes that the reason for choosing the CodeCollaborator for their implementation of lightweight peer code reviews was the positive impact it has on the Advanced Micro

Devices (AMD) as CodeCollaborator allows their internationally distributed teams to perform reviews in an accurate manner. CodeCollaborator is integrated with issue tracking and version control tools in order to enhance its efficiency when working with larger organizations. Its developers take pride in CodeCollaborator's unique ability to review user stories, test plans and user documentation along with code review. It also provides a real time chat interface. However, a short demonstration on the usage of the tool reveals that it is a centralized peer review tool which means that there is no way any one can perform reviews without the Internet [44].

### 3.2.2 Crucible

Crucible is another tool which facilitates code review. The advantages lie in the ability of sending notifications as soon as a reviewer makes changes to the code [25]. JIRA, a project tracker is integrated with Crucible to provide more flexibility to the tool. Crucible is considered to be an user friendly tool which allows generation of reports for code metrics, commit graphs and the amount of code reviewed. It is known to have over 3500 customers, including established organizations like Twitter, Adobe etc. [4]. Crucible also has a web based architecture and requires its users to be connected to the Internet while performing reviews.

### 3.2.3 Review Board

Review Board, a free and open source tool, provides its users with a powerful web based interface and command line tools for managing and simplifying the reviewing process and the review request submission. The general life cycle of performing reviews using the Review Board tool remains the same as the other tools. Review Board makes use of a tool called Django Evolution for handling any changes being made to the database. This allows updating values into the database without being bound to a database platform [13].

### 3.2.4 Rietveld

Rietveld is a code review tool which has been hosted on the Google App Engine. It is an open source version of the tool used internally by Google. Over the years, a lot of

companies and organizations have been using Rietveld. Rietveld supports Subversion development. It is used by firms like Shopify and StreamBase Systems for their internal code reviewing processes [15].

### 3.2.5 Gerrit

Gerrit, a fork of Rietveld [14], was developed by the Google software engineers who were working on Android. Gerrit has a very strong integration with Git. Pearce, the leader of the Gerrit project talks about the design decisions made while building Gerrit [43]. Unlike Git, Gerrit performs all the reviewing on a central server. People who wish to perform reviews need to go on the Gerrit website. He states that in spite of Gerrit being built on a distributed version control system Git, it allows peer to peer operation but one cannot take all the Gerrit code review data around and use it anywhere. The comments, voting information and the metadata used by Gerrit is stored in an SQL database. They would store their data in Git and let Git handle the merging.

Gerrit has a command line tool named `git-reviewed`<sup>1</sup>. This tool merely allows users to download a patch and submit a change to Gerrit. However, it does not allow the users to actually perform reviews. They would still need to go to the Gerrit website.

### 3.2.6 Bugzilla Based Review

Bugzilla is bug tracking system developed by the Mozilla project and is mainly used to keep a record of the bugs found in the project, allowing developers to work efficiently. It allows communications between the developers using Bugzilla [2]. Since it was first developed in 1998, it has been modified and different releases have ensured that it remains a user friendly tool. Bugzilla gives a lot of importance to the review process. Each patch uploaded is reviewed and comments and ratings are given by the reviewer [3]. The original purpose of Bugzilla was to report and discuss bugs, and the peer review feature was added as an afterthought.

---

<sup>1</sup>`git-reviewed` was initially named `git-review`. To avoid confusion and conflicts, we changed our tool's name to `git-reviewed`

We realize that none of the above tools are distributed in nature. All tools require reviews and discussions to be made and stored centrally.

Each tool differs from the other by having some or the other advantages, but the basic architecture of being centralized however remains the same.

### 3.3 GitHub Pull Requests

As the popularity of distributed version control systems particularly Git increased, a new dimension for distributed software development in the form of pull-based development emerged. GitHub, one of the many code hosting sites which was launched in April 2008, provided a way to perform code reviews in the form of pull requests. Pull requests are a set of commits and allow a user to let the others know about the changes pushed to a GitHub repository. After the request is sent, it is reviewed by interested parties and comments and review discussions follow. At the end of the reviewing phase, the pull request is either merged into the main repository or rejected. GitHub provides a nice way to link the code to a review and all the information about the project is stored in the GitHub repository, where people can go and browse it [16]. As of today, GitHub has over 10 million repositories, some of them owned by major software firms like Amazon, Facebook and Google [1]. Danjou [12] talks about a few problems associated with the GitHub pull-request model. This method of peer reviewing too falls into the centralized form of reviewing. All the requests are stored on a central server.

### 3.4 Email Based Code Review

Another form of lightweight code review is the email based code reviews. This type of code reviewing is preferred by most of the open source projects including Linux. The Linux kernel developer's mailing list is the most important mailbox for the Linux developers. Each day an average of 400 messages are sent on the mailing list [9]. Reviewers check the patches uploaded, comment on issues and the discussions between the author and reviewer take place via email threads.

Cohen [24] describes the advantages of an email based review. It breaks the barrier between people and anyone is free to review a code uploaded by the author. This

also gives the reviewers enough time to review any patch they want. Archives of the emails are maintained and so emails can stay around for a long time. It is very easy to implement in spite of the several disadvantages associated with the email based reviews.

Rigby *et al.* [41] explored how broadcast based reviews work on open source projects. Many patches on the mailing list are never reviewed. There are chances of biased reviewing where the patches uploaded by the renowned authors are more often reviewed. Possibilities of giving extra importance to minor issues when there is a large number of opinions floating around in a discussion are high. There is no confirmation if the patches uploaded have been reviewed with the correct solution provided. There is also no fixed amount of time for each review to complete.

Traceability is an important feature missing from the email based reviews. With tens of thousands of emails sent on the mailing list per month, it becomes extremely difficult to keep a track of so many emails. Thus, our main motive for developing git-reviewed is to keep a track of all the reviews for a particular commit. We add traceability to the process of reviewing and make it easier for the developers to go through reviews related to a commit.

## Chapter 4

# Architecture, Implementation, and Features of git-reviewed

In the previous chapter, we reviewed the literature and history of software peer review and examined the existing tools. We found that although asynchronous reviews are as effective as co-located reviews, no tool supports distributed peer review.

The requirements, description of the architecture, and the implementation of git-reviewed form the core of this chapter. We first describe the architecture of Git as git-reviewed is fully integrated with Git. We then describe git-reviewed's architecture as well as some initial failed architectural decisions. Our final architecture is drastically more elegant than our initial architecture. It consists of a single review branch (which will not affect the other branches in Git, so will not cause damage) and set of scripts that use the underlying Git API. Some of the many features provided by git-reviewed are:

- Display log history of all reviews.
- Create and send reviews as an email.
- Display interleaved history of the changes along with reviewer comments.
- Import reviews.

Furthermore, since git-reviewed does not modify Git directly, it can be seamlessly displayed in other environments that developers might be familiar with, such as GitHub.



## 4.1 High-level Features for git-reviewed

git-reviewed was built keeping in mind the design goals we intended to achieve. We wanted to develop a tool which was highly distributed, lightweight and efficient. We describe our design goals below:

### **Lightweight**

Almost all of the modern code review tools today are lightweight. They provide an environment where many developers can work together. Lightweight review tools also allow extensive knowledge sharing. git-reviewed facilitates lightweight project development methodologies. Reviews can be transferred from one developer personal repository to another developer's repository, allowing them to work simultaneously, yet in a distributed manner. Moreover, git-reviewed has no dependencies on any external systems. It does not rely on storing the reviews in any particular database. git-reviewed is highly robust and has an ability to store over 100K reviews as already present for the main repository of Linux. There are no extra system requirements and it is a very stable tool.

### **Distributed**

git-reviewed is distributed in nature and it does not rely on any centralized server. No code review tool which is distributed is available. git-reviewed maximizes the advantages a distributed version control system provides. It means that traceability can be added to the process of reviewing without going on any centralized server to view your reviews and discussions. Everything is present locally.

### **Usability**

git-reviewed is targeted to be as simple as possible. Our goal was to not change the current reviewing ethics of the developers and simultaneously add more functionality to it. The current developers using code review tools especially the Linux developers were our target audience.

## Efficiency

git-reviewed is efficient and quickly returns results to the users, thereby, encouraging the adoption of the tool. The tool provides accurate information and allows users to make reviews and push them to a separate branch dedicated only to store reviews efficiently in no time. git-reviewed stores all the reviews on the ‘review’ branch ordered by date. This makes it easy for the users to go through the most recent reviews first. Since all the reviews are stored on a separate branch, there are no chances of them getting lost.

## 4.2 Git Architecture

git-reviewed is built on top of Git and extensively uses the Git API. Therefore, it is important to explain the internal details and the architecture of Git. Git was fully distributed and had support for non-linear development which was facilitated by the branching feature. Git could efficiently handle a large project in terms of speed and data size [6]. Git stores its data as a series of snapshots of the project and uses a checksum mechanism which stores everything as an object which can be addressed by a SHA-1 hash value of its contents.

Once any Git repository is initialized, the objects directory is automatically created which stores all the contents as a single file named with the SHA-1 hash value of its content and header.

Git has four different types of objects and every type of content in the Git repository is one of these objects. Figure 3 shows the four types of objects of the Git architecture.

### Blob

A blob represents a single file stored in the repository. A blob is a collection of bytes which can either be a text file or a binary file etc. Whenever a file is added to the Git repository, a hash value is computed and it is stored in the .git directory. Since SHA-1 hash is statistically, globally independent, the contents stored in the file will be unique. Any other file with the same contents will not be allowed to be stored as the same hash value will be computed and Git stores only one copy of the file.

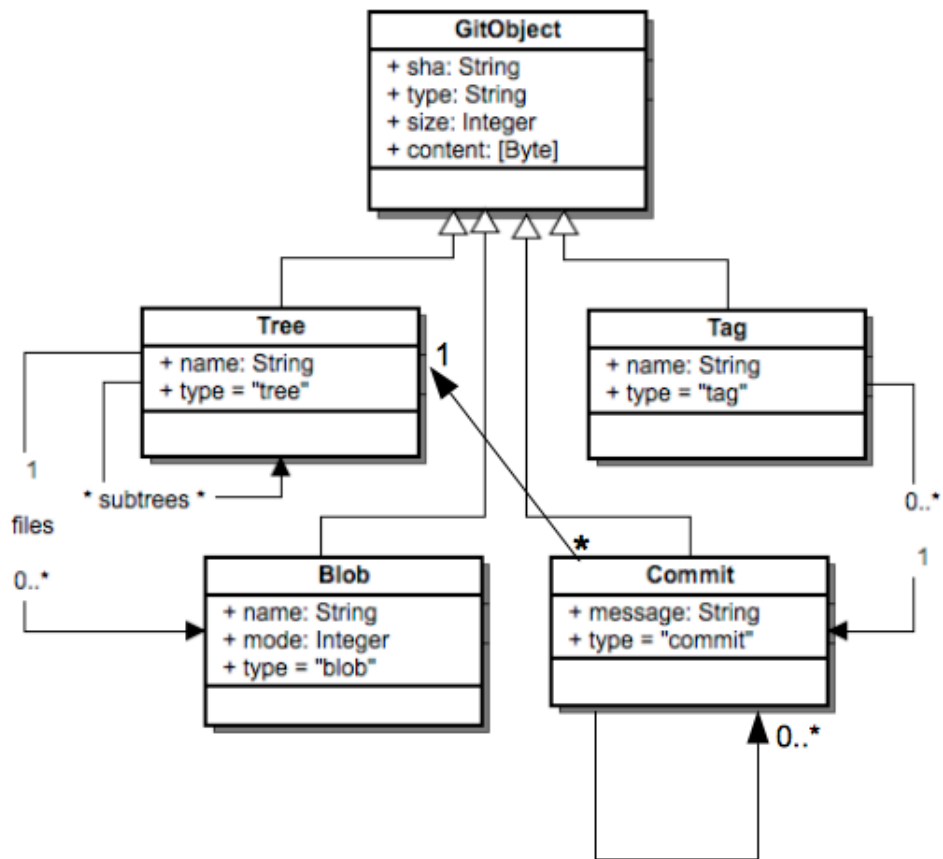


Figure 3: Git Architecture, modified from [34]

## Tree

A tree object acts like a directory which stores the information about the file names and also helps us to group a collection of files. A single tree object can store one or more trees or a blob object with its associated mode, type, and file name. Git also allows us to create our own trees.

## Commit

A commit object stores a snapshot of the contents of the repository at the time they were committed. A commit object points to the top level tree and the parent commit. It saves all the information and hash values of all the objects need not be remembered.

## Tag

A tag object stores a name and it points to the commit object for the time the tag object represents. It is normally used to give a commit a name. If no particular commit is specified, by default Git tags the most recent commit.

### 4.2.1 Branching

Git's branching model is the most important feature which sets it apart from most of the version control systems. It is extremely lightweight and makes Git a powerful tool. The branching and merging mechanism allows a developer to diverge from the current module he is working on by creating a new branch, make changes on it and when satisfied, the work can be merged to the original work. A branch allows a line of development to be isolated, so that changes made to the current working model don't affect it and vice versa.

Barr *et al.* [19] conducted a study which found answers to the popularity of the distributed version control systems. Through their interviews and the analysis with the project leaders of the open source projects, they found out that the main reason for developers to adopt the distributed version control systems was the ability of the branching mechanism from protecting the developers from any kind of interruptions by providing isolation. They also found out that branches were more cohesive than the background commit sequences.

## 4.2.2 git-reviewed Integration with Git

git-reviewed is fully integrated with Git and uses the standard Git commands. git-reviewed allows users to see all the reviews relevant to each commit on the Git repository. Users can perform reviews for a commit on a Git repository by following the steps given below:

1. Clone the Git repository.
2. Install git-reviewed.
3. A review branch gets created as soon as you start reviewing.
4. Perform reviews, make comments.
5. Push the changes from the local repository onto the origin.

## 4.3 Initial Architecture of git-reviewed

We explain the architecture of git-reviewed in this section. There have been several changes to the architecture since we started designing git-reviewed. The initial architecture failed due to the restrictions imposed by Git and so we had to design a new architecture which did not break the way Git worked and also allowed us to implement a completely distributed tool. The design trade-off, failures of the initial architecture and the current architecture are discussed in detail in the following subsections.

### 4.3.1 Initial Architecture of git-reviewed

In order to meet with our design goals, a lot of research went into incorporating the distributed peer reviewing tool into the current Git architecture. We had to make sure that minimal changes had to be made as it had to maintain the reviewing practices of the current developers.

#### Review Objects

We initially decided to create customized review objects. These review objects would in turn contain information of any reviews created in response to them. It would

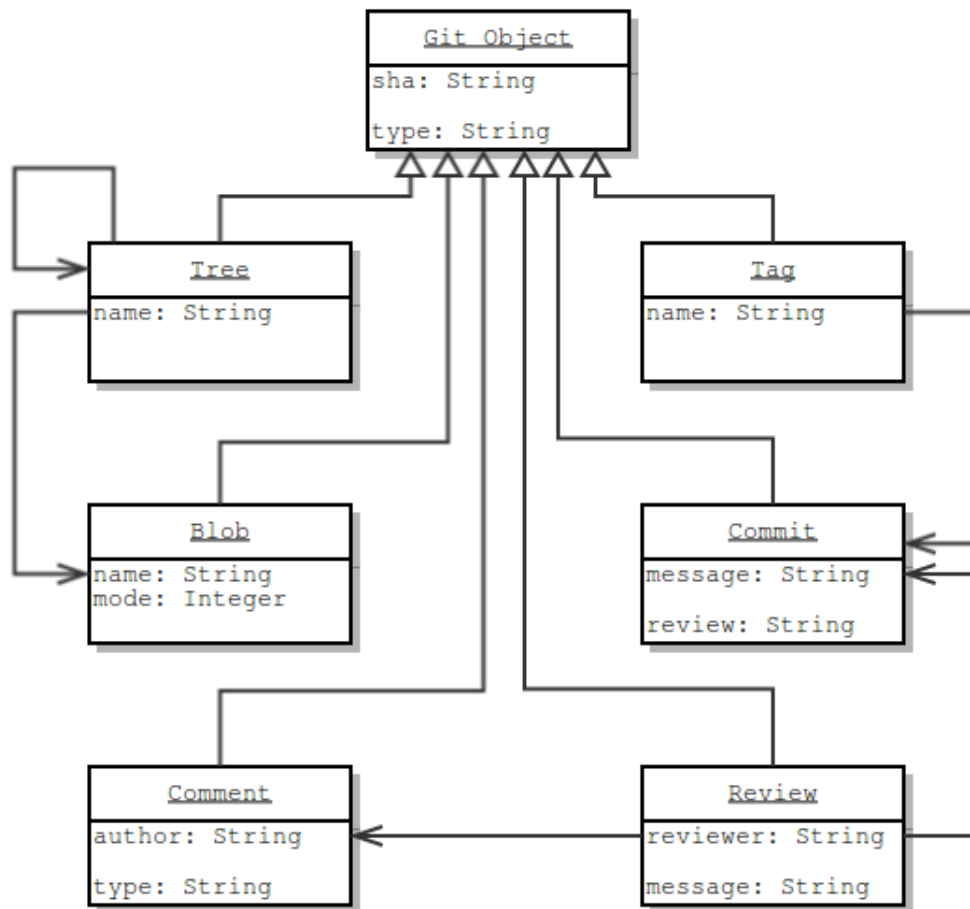


Figure 4: Initial architecture of git-reviewed

also have the information about the comment objects that were specifically created for these review objects. When any commit was reviewed using `git-reviewed`, a new review object would be created which would consist information about the commit being reviewed.

## Comment Objects

When any comment was made on the review object, a comment object would get created. This comment object would consist of the actual comments made by the user, line number on which the comment was made and information about any other comment objects created for that review.

This proposed architecture as shown in Figure 4 failed as Git did not allow users to create customized Git objects. Figure 5 shows how the current Git architecture could not recognize custom objects, a review object in this case. An object could only be one of the types as explained in Section 4.2. We had to implement `git-reviewed` in a way such that reviews, comments would all be stored in the Git repository and the underlying architectural design of Git was not violated. It would have been possible to modify Git, but then the users would have to install our modified version of Git.

```
murtuza@murtuza-ThinkPad-X230:~/murtuza$ git cat-file -t 1d154cd6d692ceea4cc7d6a7f8f087f8566343e0
fatal: invalid object type "review"
```

Figure 5: Invalid object error in Git

### 4.3.2 Current Architecture of `git-reviewed`

Figure 6 shows the architectural diagram of `git-reviewed`. We store all the reviews on a branch named the ‘review’ branch. This branch does not share a root node with the master branch and is a free standing branch which points to the last review committed. The advantage of having all reviews stored on a separate branch is that reviews do not interfere with the developer’s view of the system history. Figure 7 shows an instance of the architectural diagram. We can see how the commit folders

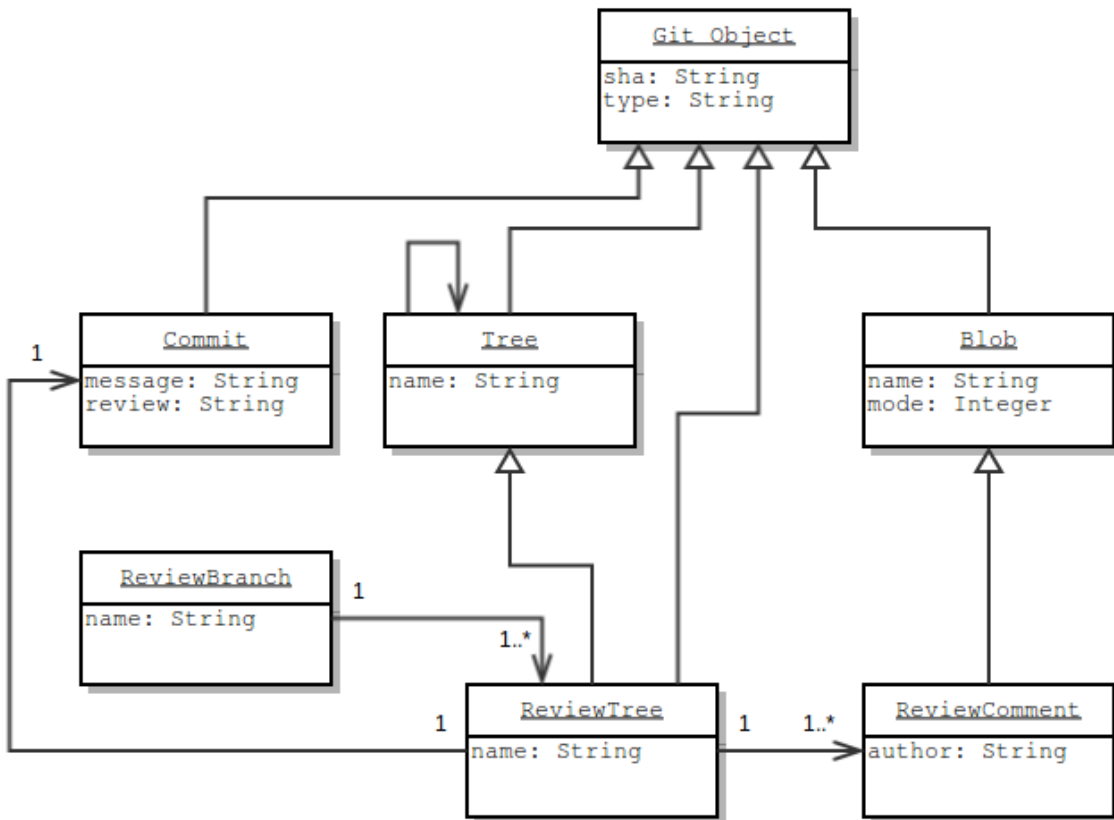


Figure 6: Architectural diagram of git-reviewed



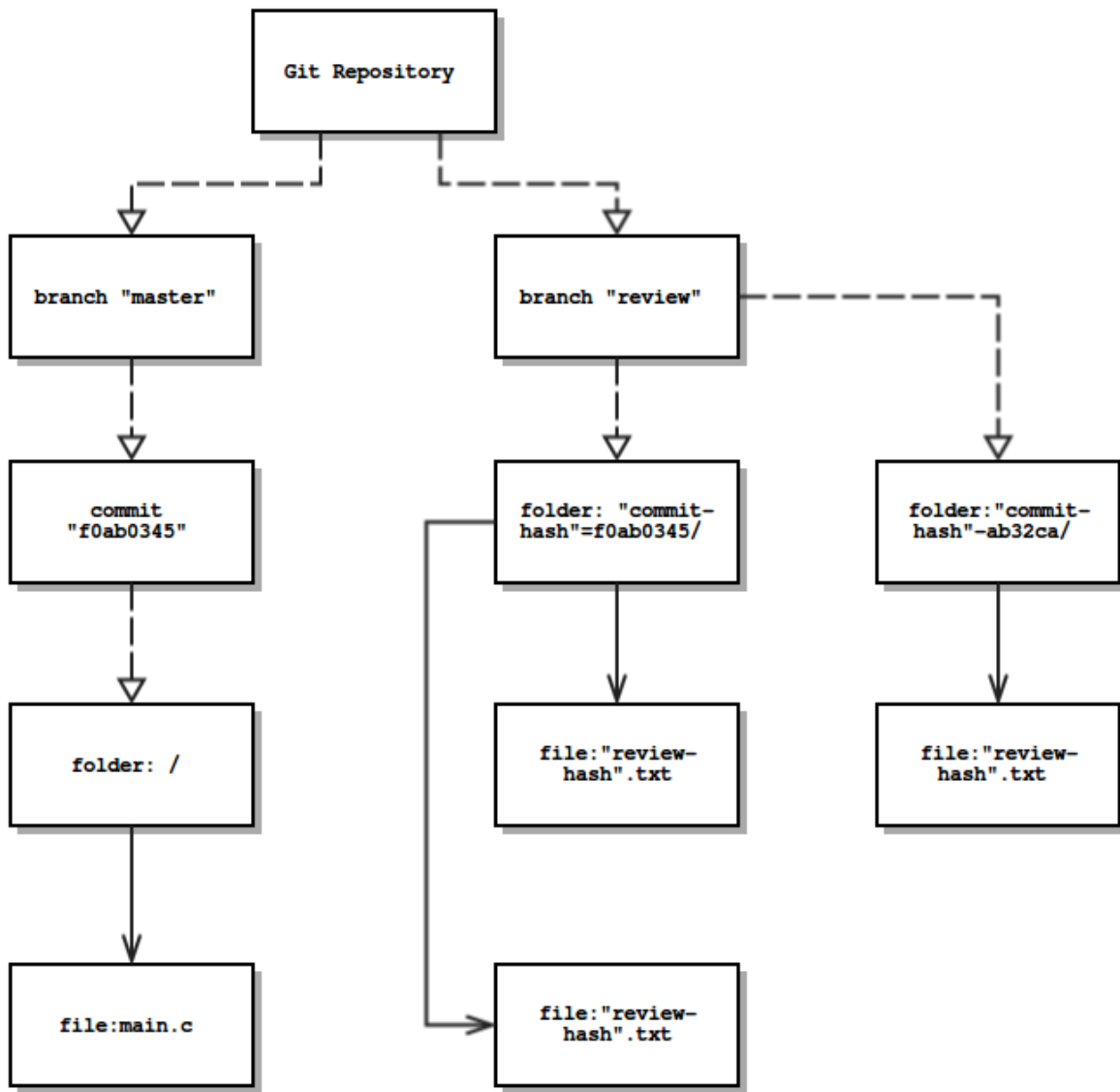


Figure 7: An instance of git-reviewed

and the review objects are organized on the local machine.

## **Review Branch**

git-reviewed takes advantages of the branching model of Git. It stores all reviews on a separate detached branch called the ‘review’ branch. As soon as the user installs git-reviewed and creates the first review, this branch gets automatically created. The risk with this branch is minimal as it does not interfere with the master branch or the other branches present on the Git repository.

## **Reviews**

All the reviews are stored on the dedicated ‘review’ branch as blob objects. The review object like every other object in the repository is identified by a unique SHA-1 hash. Since the reviews are just blobs stored on the branch like any other blob, it has an access mode and is stored in the tree. All the comments made by users for the commit would be stored in these review blob objects.

## **Commit Folders**

All the review blobs are stored in the tree. We name this tree after the commit-hash on the master branch for which the review has been made. This helps us in storing all the reviews for that particular commit in one tree. We can have multiple reviews per commit and we can also maintain a commit-review index. As soon as a new review is made, all the reviews are committed and the commit refers to the old commit as its parent. The main advantage of having this architecture as we can transmit reviews between repositories, most importantly allowing us to merge reviews.

## **4.4 Implementation of git-reviewed**

git-reviewed is developed using a set of Ruby and Perl scripts consisting of over 2000 lines of code. These scripts make use of the current Git API, which help in creating and storing reviews and discussions. git-reviewed also provides functionalities, which assist users in dealing and working with reviews.

The different modules of this tool get executed depending on the type of command entered the user. There are a total of 21 functions currently which is performed by git-reviewed. A detailed list of all the functions can be found in Appendix [B.2](#).

We separately explain the different modules of git-reviewed in order to easily understand the overall implementation details of our tool.

#### **4.4.1 git-reviewed Option Parser**

This is the first module which is executed as soon as a user types in a command in git-reviewed. Using Ruby's command line option parser, we determine which call to make in order to execute the function requested by the user. This also determines if a new review has to be created or a review is a threaded response to another review.

#### **4.4.2 Communication with Remote Repository**

This module of git-reviewed takes care of all the communication that takes place between the local repository and the remote repository. Reviews are pulled and pushed using this module and it also plays an important role in the initial set up required to use git-reviewed on the local machine efficiently.

#### **4.4.3 Creating and Deleting Reviews**

Creating and deleting reviews is handled by a separate part of our tool, which makes sure that reviews are stored in the correct commit folder. It also checks if a review has been previously created and does not allow duplicate reviews to be stored. It also is designed to handle various merge conflicts which may arise while performing reviews.

#### **4.4.4 Creating and Sending Emails from git-reviewed**

git-reviewed creates a patch file from the commit or a set of commits just like Git. One can import or export emails to the Linux kernel mailing list directly from the tool.

### 4.4.5 Developing Log History of Reviews

This module allows users to view the log history of the reviews stored on the repository. It also allows users to view the log of reviews for a particular commit or view reviews made by a particular reviewer.

In the next section, we show the different features and the functionalities git-reviewed has to provide.

## 4.5 Features and Functionality

In the previous section, we described how our architecture allows us to implement the following features:

- Allow storage of multiple reviews per commit
- Maintain a commit-review index
- Allow merging of reviews
- Allow transfer of reviews through various repositories.

In each of the following subsections, we give an example of how the feature is used, relate it to our architecture, and to the needs of the Linux community.

In order to make git-reviewed more usable, we had to add more features and functionality apart from just creating reviews and making comments in each review. We studied the functionalities Git provides and tried to implement it in our tool. git-reviewed user commands included ways to locate each review on the review branch and add more traceability.

Some of the features of git-reviewed are described below:

### 4.5.1 Viewing a List of all Reviews

git-reviewed allows displaying a list of all reviews which are currently stored on the 'review' branch of a repository. `git review - -list` command shows the number of reviews, the review hash as well as the commit reviewed. Figure 8 shows the command line interface and how reviews will be listed on git-reviewed.

```
Total Number of Reviews: 148

Review: 413e914f466424b587824c51c3db511637f8f823
Commit Reviewed: 0015e551edb1d28191567d8a7d1ce5edda404ced

Review: 5eac18c959052a4204c59e91c2fdedc097821ca6
Commit Reviewed: 0015e551edb1d28191567d8a7d1ce5edda404ced

Review: 7c438386297dff00db43955a2b9a3892077d998f
Commit Reviewed: 0343c5543b1d3ffa08e6716d82afb62648b80eba

Review: 8612f96d240a145c83b8b6d7c4078c000c74be2e
Commit Reviewed: 058bd4d2a4ff0aaa4a5381c67e776729d840c785

Review: d26b1c977d446f0865ec911d240115e74f5be1cc
Commit Reviewed: 058bd4d2a4ff0aaa4a5381c67e776729d840c785

Review: 425ec01fcabc9bb5d1bd909893b691b2c6b2f663
Commit Reviewed: 06d6c1087605b38342eb20e74b0cacb8b71f5086

Review: 971d3c37c83ee0719605684ed9ac8e4474742a2c
Commit Reviewed: 06d6c1087605b38342eb20e74b0cacb8b71f5086

Review: d0374d0b5fba4d16c1c117263540150238cd73c8
Commit Reviewed: 06d6c1087605b38342eb20e74b0cacb8b71f5086

Review: e08c7204bec7e80b5c5b5d0131eb7d45d9dde68
Commit Reviewed: 06d6c1087605b38342eb20e74b0cacb8b71f5086

Review: 8bfa4ee1502b3972c543057b55d0eb3e83a92458
Commit Reviewed: 09009512e5e7ab341b1554a256f81dd512c1f4bf

Review: 44839c22a6eb345ff6190741b99620db95646b53
Commit Reviewed: 0a282538cc1977655004cdb2eb25dd2b63f20637

Review: 9b9810a94ef317c747ededc0f8a6c2b8716617eb
Commit Reviewed: 0a282538cc1977655004cdb2eb25dd2b63f20637

Review: 48ec97debb48efe95baca820d449ce49523978d5
Commit Reviewed: 0f09a343dabeadc4dcde4714317e25de2e93fd13
:
```

Figure 8: git-reviewed displaying the list of reviews

## 4.5.2 Log of Reviews

git-reviewed also shows log of all the reviews which keep a track of time, reviewer details and the commit details for each review. Figure 9 shows how the log of all reviews ordered by date are displayed on git-reviewed.

## 4.5.3 Viewing Each Review

All the email discussions that take place on the mailing list is converted into a review. Each review consists of the comments and the messages that were sent on the mailing list. The content of each review is stored exactly how the message is stored on the mailing list. Figure 10 shows the details of each review.

## 4.5.4 Viewing Reviews Related to a Particular Reviewer

git-reviewed allows a user to view the reviews related to a particular reviewer. This is useful when there are a large number of reviews and only interesting reviews are needed to be displayed. Figure 11 shows the reviews only for a particular reviewer.

## 4.5.5 Creating a Review

git-reviewed allows users to create their own reviews on each commit and uploading the patch on the mailing list. Figure 12 shows how a text editor pops up which contains the commit details and allows the reviewer to review the commit.

## 4.5.6 Importing Reviews from a Mailing List

To ensure a smooth transition for the developers from the current process of reviewing to performing reviews using git-reviewed, we made sure that git-reviewed allows developers to import reviews from the mailing list and also provided them their own public repositories that already consisted of recent reviews they had been working on.

When we sent out links to the GitHub repositories to the OSS developers to perform an evaluation of our tool (see Section 5), we had over 200K reviews stored on the review branches of these repositories from January 2012 onwards.

```
Review: 3bfb4f3c0e16869616deccbe707787f9a40b858a
Commit Reviewed: 1622ac23bd3568c3ae8bb391dd3adb51887d7141
Author: Ian Campbell <ijc@hellion.org.uk>
AuthorDate: Mon Feb 4 16:47:56 2008 +0100
Reviewer: Ian Campbell ijc@hellion.org.uk <ijc@hellion.org.uk>
ReviewDate: Fri Feb 1 10:58:14 2008 +0000

Re: [PATCH] x86: add a crc32 checksum to the kernel image.

Review: 2b0ccc0a85a2ce5cb5796166febe5c5f5c562714
Commit Reviewed: 1622ac23bd3568c3ae8bb391dd3adb51887d7141
Author: Ian Campbell <ijc@hellion.org.uk>
AuthorDate: Mon Feb 4 16:47:56 2008 +0100
Reviewer: Ian Campbell ijc@hellion.org.uk <ijc@hellion.org.uk>
ReviewDate: Fri Feb 1 09:02:48 2008 +0000

Re: [PATCH] x86: add a crc32 checksum to the kernel image.

Review: ef82ce5bd6ded1221922ab2770e05ca68725bb92
Commit Reviewed: 1622ac23bd3568c3ae8bb391dd3adb51887d7141
Author: Ian Campbell <ijc@hellion.org.uk>
AuthorDate: Mon Feb 4 16:47:56 2008 +0100
Reviewer: David Newall davidn@davidnewall.com <davidn@davidnewall.com>
ReviewDate: Fri Feb 1 16:16:06 2008 +1030

Re: [PATCH] x86: add a crc32 checksum to the kernel image.

Review: 4cae4a178e9766d48ca31308bbbd6a5daa67df05
Commit Reviewed: 1622ac23bd3568c3ae8bb391dd3adb51887d7141
Author: Ian Campbell <ijc@hellion.org.uk>
AuthorDate: Mon Feb 4 16:47:56 2008 +0100
Reviewer: David Newall davidn@davidnewall.com <davidn@davidnewall.com>
ReviewDate: Fri Feb 1 11:13:27 2008 +1030

Re: [PATCH] x86: add a crc32 checksum to the kernel image.
: |
```

Figure 9: git-reviewed displaying the log of reviews

```

Commit Reviewed: d024206133ce21936b3d5780359afc00247655b7
Reviewer: Luis Henriques <luis.henriques@canonical.com>
Reviewer Date: Fri, 7 Feb 2014 11:46:43 +0000
-----
From : Luis Henriques <luis.henriques@canonical.com>
commit: d024206133ce21936b3d5780359afc00247655b7
Message-Id : <1391773652-25214-185-git-send-email-luis.henriques@canonical.com>
Date : Fri, 7 Feb 2014 11:46:43 +0000
Subject: [PATCH 3.11 184/233] btrfs: restrict snapshotting to own subvolumes
References: <1391773652-25214-1-git-send-email-luis.henriques@canonical.com>
In-Reply-To: <1391773652-25214-1-git-send-email-luis.henriques@canonical.com>
Cc: David Sterba <dsterba@suse.cz>, Josef Bacik <jbacik@fb.com>, Chris Mason <clm@fb.com>, Luis Henriques
3.11.10.4 -stable review patch. If anyone has any objections, please let me know.
-----

From: David Sterba <dsterba@suse.cz>

commit d024206133ce21936b3d5780359afc00247655b7 upstream.

Currently, any user can snapshot any subvolume if the path is accessible and
thus indirectly create and keep files he does not own under his directories.
This is not possible with traditional directories.

In security context, a user can snapshot root filesystem and pin any
potentially buggy binaries, even if the updates are applied.

All the snapshots are visible to the administrator, so it's possible to
verify if there are suspicious snapshots.

Another more practical problem is that any user can pin the space used
by eg. root and cause ENOSPC.

Original report:
https://bugs.launchpad.net/ubuntu/+source/apparmor/+bug/484786

Signed-off-by: David Sterba <dsterba@suse.cz>
Signed-off-by: Josef Bacik <jbacik@fb.com>
Signed-off-by: Chris Mason <clm@fb.com>
[ luis: backported to 3.11: adjusted context ]
Signed-off-by: Luis Henriques <luis.henriques@canonical.com>
---
 fs/btrfs/ioctl.c | 6 +++++
 1 file changed, 6 insertions(+)

diff --git a/fs/btrfs/ioctl.c b/fs/btrfs/ioctl.c
index 6074a8e..6752118 100644
--- a/fs/btrfs/ioctl.c
+++ b/fs/btrfs/ioctl.c
@@ -1534,6 +1534,12 @@ static noinline int btrfs_ioctl_snap_create_transid(struct file *file,
     printk(KERN_INFO "btrfs: Snapshot src from "
             "another FS\n");
     ret = -EINVAL;
+
+ } else if (!inode_owner_or_capable(src_inode)) {
+     /*
+      * Subvolume creation is not restricted, but snapshots
+      * are limited to own subvolumes only

```

Figure 10: Viewing each review on git-reviewed



```
Review: f415d9671e2cb4f716603193fce154b6ace7ab79
Commit Reviewed: 9e58d05a1b24d2c0471c3b4df8f473a7543d7647
Author: Samuel Iglesias Gonsálvez <siglesias@igalia.com>
AuthorDate: Fri Jul 20 09:39:03 2012 +0200
Reviewer: Greg Kroah-Hartman gregkh@linuxfoundation.org <gregkh@linuxfoundation.org>
ReviewDate: Fri Dec 14 15:00:43 2012 -0800

[ 06/37] Staging: ipack/bridges/tpci200: avoid kernel bug when uninstalling a device

Review: bc5403e1a722eca0f8c283c6a825696f41e43384
Commit Reviewed: 55220bb3e5f917dd5fee1153c612f9a83599f639
Author: Florian Fainelli <florian@openwrt.org>
AuthorDate: Mon Dec 10 12:25:32 2012 -0800
Reviewer: Greg Kroah-Hartman gregkh@linuxfoundation.org <gregkh@linuxfoundation.org>
ReviewDate: Fri Dec 14 15:00:44 2012 -0800

[ 07/37] Input: matrix-keymap - provide proper module license

Review: ccad604e25e30a49367ed909eaae115db1401390
Commit Reviewed: 878d7439d0f45a95869e417576774673d1fa243f
Author: Eric Dumazet <edumazet@google.com>
AuthorDate: Thu Oct 18 04:55:36 2012 -0700
Reviewer: Greg Kroah-Hartman gregkh@linuxfoundation.org <gregkh@linuxfoundation.org>
ReviewDate: Fri Dec 14 15:01:13 2012 -0800

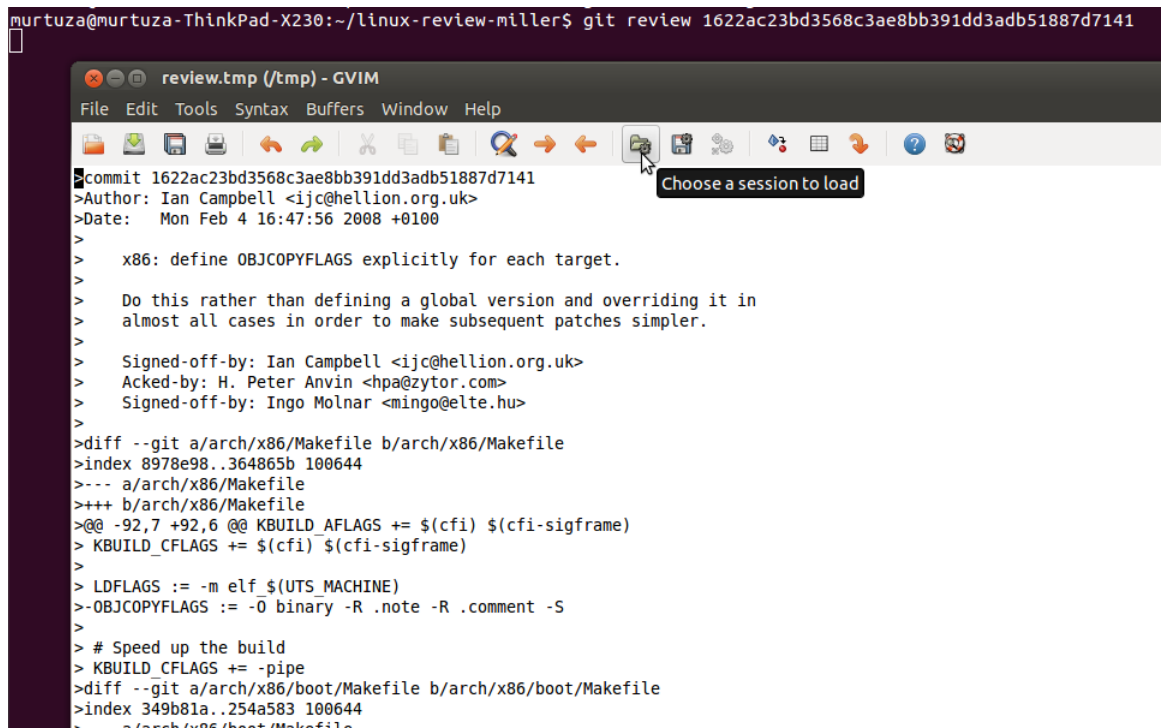
[ 36/37] rcu: Fix batch-limit size problem

Review: 01bc76043ae4177841b2777e09c2a4b5c432cb83
Commit Reviewed: 129ff8f8d58297b04f47b5d6fad81aa2d08404e1
Author: Zhang Rui <rui.zhang@intel.com>
AuthorDate: Tue Dec 4 23:30:19 2012 +0100
Reviewer: Greg Kroah-Hartman gregkh@linuxfoundation.org <gregkh@linuxfoundation.org>
ReviewDate: Fri Dec 14 15:01:03 2012 -0800

[ 26/37] ACPI / video: ignore BIOS initial backlight value for HP Folio 13-2000
:█
```

Figure 11: Viewing reviews for a particular reviewer on git-reviewed

```
murtuza@murtuza-ThinkPad-X230:~/linux-review-miller$ git review 1622ac23bd3568c3ae8bb391dd3adb51887d7141
```



```
review.tmp (/tmp) - GVIM
File Edit Tools Syntax Buffers Window Help
Choose a session to load
commit 1622ac23bd3568c3ae8bb391dd3adb51887d7141
>Author: Ian Campbell <ijc@hellion.org.uk>
>Date: Mon Feb 4 16:47:56 2008 +0100
>
> x86: define OBJCOPYFLAGS explicitly for each target.
>
> Do this rather than defining a global version and overriding it in
> almost all cases in order to make subsequent patches simpler.
>
> Signed-off-by: Ian Campbell <ijc@hellion.org.uk>
> Acked-by: H. Peter Anvin <hpa@zytor.com>
> Signed-off-by: Ingo Molnar <mingo@elte.hu>
>
>diff --git a/arch/x86/Makefile b/arch/x86/Makefile
>index 8978e98..364865b 100644
>--- a/arch/x86/Makefile
>+++ b/arch/x86/Makefile
>@@ -92,7 +92,6 @@ KBUILD_AFLAGS += $(cfi) $(cfi-sigframe)
> KBUILD_CFLAGS += $(cfi) $(cfi-sigframe)
>
> LDFLAGS := -m elf_${UTS_MACHINE}
>-OBJCOPYFLAGS := -O binary -R .note -R .comment -S
>
> # Speed up the build
> KBUILD_CFLAGS += -pipe
>diff --git a/arch/x86/boot/Makefile b/arch/x86/boot/Makefile
>index 349b81a..254a583 100644
>--- a/arch/x86/boot/Makefile
```

Figure 12: Creating a review on git-reviewed

In order to gather maximum feedback for our tool, we started looking into other potential projects which git-reviewed could track reviews for. We realized that the reviewing practices followed by teams for other OSS projects (e.g. PostgreSQL and Git) were the same (email based code reviewing), and so, we started tracking the reviews from the PostgreSQL and Git mailing lists respectively.

PostgreSQL is an open source object-relational database and has been a leading database system in the market for the past 15 years. It runs on all major platforms and gives a lot emphasis on extensibility and standards-compliance [11].

The development of the distributed version control system, Git, is similar to the Linux kernel development as it was built by the same community of developers. Git is now maintained by Junio Hamano. Git has its own mailing list and since git-reviewed is extensively embedded into Git, we wanted to allow the Git developers to use git-reviewed while performing their reviewing tasks.

We subscribed to the mailing lists for Linux, Git and PostgreSQL, and continued to collect the mails sent on these lists on a daily basis. A script which pulled

these messages from the subscribed email account and stored it into mbox files was implemented. Using the `git-review-update` command we could create reviews from the messages stored in these mbox files and store them on the ‘review’ branch on the respective Git repository for each project. This feature would not restrict the users of `git-reviewed` to work with the reviews created and hosted by us on a daily basis. They could also create reviews using their own mbox files using the same command.

### 4.5.7 GitHub Compatibility

Viewing reviews on GitHub is easy as each review is stored in a folder named as the commit for which the review has been made. This organizes the reviews very well and provides a good graphical user interface for the reviews as shown below. You can see how reviews are organized on the review branch.

Figure 13 shows the list of commit folders for the commits which have been reviewed at least once. Figure 14 and 15 show the list of reviews made in each commit on GitHub and the contents of the review respectively.

#### Repositories on GitHub

The reviews are extracted from the Linux kernel, Git and the PostgreSQL mailing lists daily. The reviews are hosted on the following GitHub repositories which are available for browsing. We have matched the commits and reviews from the year 2012 onwards for the Linux, Git and the PostgreSQL projects as well as personalized Linux subsystem maintainer’s repositories.

1. <https://github.com/mmukadam/linux.git>
2. <https://github.com/mmukadam/linux-review-v3.12.git>
3. <https://github.com/mmukadam/kernel-git-davem-net.git>
4. <https://github.com/mmukadam/kernel-git-gregkh-usb.git>
5. <https://github.com/mmukadam/kernel-git-tiwai-sound.git>
6. <https://github.com/mmukadam/kernel-git-tj-libata.git>

branch: review linux-review-miller /

This branch is 145 commits ahead and 400357 commits behind master [Pull Request](#) [Compare](#)

Re: pull request: wireless 2012-10-31

Johannes Berg authored a year ago latest commit 78307aeb11  
 → davem330 committed a year ago

0115e8e30d6fcdd4b8faa30d3...	Re: [patch net-next v2 01/15] net: introduce upper device lists	a year ago
043c4789726e6abb5be6115c...	Re: [PATCH 17/20 V2] drivers/net/ethernet/sun/niu.c: fix error return...	a year ago
06e3041164a1c52d6e8a369a...	Re: [PATCH net-next] pktgen: Use ipv6_addr_any	a year ago
0e698bf6624c469cd4f3f912...	Re: [PATCH] net: fix memory leak on oom with zerocopy	a year ago
0f48917b93414a9c306a834b...	Re: [PATCH net-next, 1/1] hyperv: Add comments for the extended buffer...	a year ago
14834540caff24944affd6290f...	Re: [PATCH 12/20 V2] drivers/net/ethernet/qlogic/qlcnic/qlcnic_main.c...	a year ago
155e4e12b9f49c2dc817bb4c...	Re: [PATCH] batman-adv: Fix mem leak in the batadv_tt_local_event() f...	a year ago
15b9350a177b9fb23b69e00f...	[PATCH v2 1/7] sparc64: Only support 4MB huge pages and 8KB base pages.	a year ago
16c0b164bd24d44db137693...	Re: [net V2] act_mirred: do not drop packets when fails to mirror it	a year ago
16ebd60856bc5d980722cb8...	Re: pull request: wireless 2012-08-03	a year ago
16fa9e1d104e6f2c180054ac...	Re: [PATCH][v2] netdev/phy: mdio-mux-mmio.c should include of_addr...	a year ago
2120c52da6fe741454a60644...	Re: Endianess bug fix for sierra_net	a year ago
249ee72249140fe5b9adc988f...	Re: pull request: wireless 2012-10-09	a year ago
27f011243a6e4e8b81078df1d...	Re: pull request: wireless 2012-08-24	a year ago
28407630513b1a86133db0ef...	<a href="#">Re: [PATCH v2 01/15] net: proc entry showing inodes on sockfs and the...</a>	a year ago
2a6c8c7998f95b140f3d3c7ac...	Re: [PATCH] net: scm: moving dereference after NULL check	a year ago
2c9693222952b08c224093a...	Re: linux-next: build failure after merge of the final tree (net-next...	a year ago
381c726c09bb43aea8088ed...	Re: pull request: wireless 2012-09-07	a year ago
39707c2a3ba5011038b363f8...	Re: [PATCH] net: usb: Fix memory leak on Tx data path	a year ago
3a40414f826a8f1096d9b94c4...	Re: pull request: wireless 2012-10-19	a year ago
3a7c384ffd57ef5fbd95f48eda...	Re: [PATCH] ipv4: tcp: unicast_sock should not land outside of TCP stack	a year ago

Figure 13: Commits reviewed on GitHub

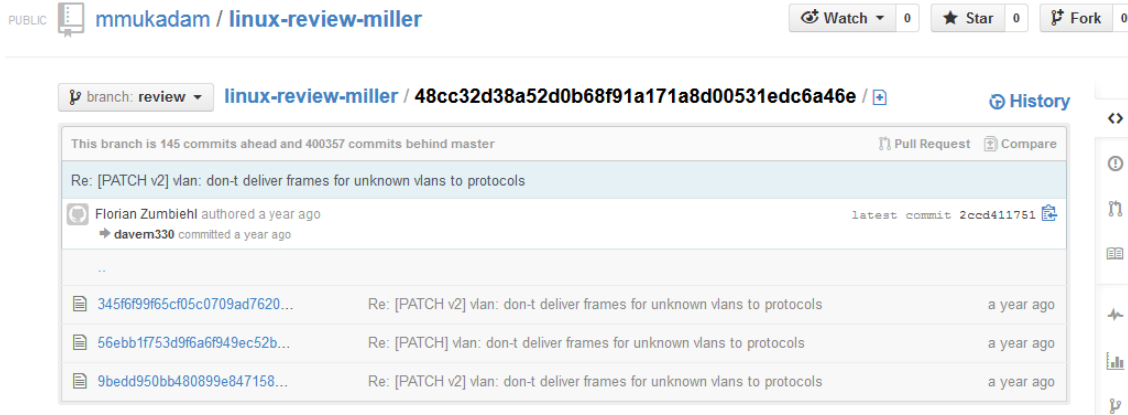


Figure 14: Reviews for a specific commit on GitHub

7. <https://github.com/mmukadam/kernel-git-rafael-linux-pm.git>
8. <https://github.com/mmukadam/postgres.git>
9. <https://github.com/mmukadam/git.git>

## 4.6 Need of Linux Reviewers and Features of git-reviewed

The reviewers on the Linux kernel have been performing reviews for more than a decade. They have adopted the mailing list and perform over 400 reviews per month [36]. Our task was to implement features into git-reviewed which would make sure that the developers do not really have to change their reviewing habits in order to use git-reviewed. We have made the following contributions in making our tool very similar to the developers' needs:

### Store Reviews in Email Quoted Manner

The format of the reviews created from the messages on the mailing list is kept intact. We also make sure that the new reviews created by developers are all formatted in an email quoted manner. This makes it easier for the developers as they can continue reviewing just the way they have been for the past decade.

branch: review ▾

[linux-review-miller / 48cc32d38a52d0b68f91a171a8d00531edc6a46e /](#)  
[345f6f99f65cf05c0709ad762088cc2ffdb692b9](#)

Florian Zumbiehl a year ago Re: [PATCH v2] vlan: don't deliver frames for unknown vlans to protocols

0 contributors

file | 37 lines (29 slocc) | 1.622 kb

Open Edit Raw Blame History Delete

```

1 From : David Miller <davem@davemloft.net>
2 Message-Id : <20121008.144230.1404596032615788891.davem@davemloft.net>
3 Date : Mon, 08 Oct 2012 14:42:30 -0400 (EDT)
4 Subject: Re: [PATCH v2] vlan: don't deliver frames for unknown vlans to protocols
5 In-Reply-To: <20121008015158.GE25895@florz.florz.dyndns.org>
6 References : <20121008015158.GE25895@florz.florz.dyndns.org>
7 Cc: kaber@trash.net, eric.dumazet@gmail.com, netdev@vger.kernel.org, linux-kernel@vger.kernel.org, jpirko@redhat.com
8 commit: 48cc32d38a52d0b68f91a171a8d00531edc6a46e
9
10 From: Florian Zumbiehl <florz@florz.de>
11 Date: Mon, 8 Oct 2012 03:51:58 +0200
12
13 > This version completely avoids any new state that could need to be spilled
14 > to RAM, and instead re-checks existence and non-zerosness of the tag. What
15 > do you think?
16
17 At a high level it looks fine and doesn't have the problems mentioned
18 earlier.
19
20 But I wonder if it breaks things, since you do the assignment so late
21 we no longer handle the case where the VLAN device's MAC address
22 matches the packet MAC address and the top-level device's does not.
23
24 That's handled by logic in vlan_do_receive() which checks for
25 PACKET_OTHERHOST.
26
27 But you're going to unconditionally set PACKET_OTHERHOST, overriding
28 any decision that code makes.
29
30 This turns out to be a really non-trivial area and it's going to take
31 some time to get this right and audit the change appropriately.
32 --
33 To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
34 the body of a message to majordomo@vger.kernel.org
35 More majordomo info at http://vger.kernel.org/majordomo-info.html
36 Please read the FAQ at http://www.tux.org/lkml/

```

Figure 15: Review on GitHub

## Threading of Reviews

git-reviewed allows threading of reviews. All the review responses are stored in the same commit folder and they consist information about the thread header. This features allows a review thread to be generated just the way developers are used to while reviewing on the mailing list.

## Import of Emails

git-reviewed allows import of emails. As explained in Section 4.5.6, git-reviewed's feature of importing emails encourages adoption of the tool.

## Commands Similar to Git

We tried to keep the commands for git-reviewed very similar to the Git commands. A comparison of Git and git-reviewed commands are shown in Table 2.

## Sending Emails Directly from git-reviewed

git-reviewed allows developers to directly prepare a review for email submission and send it to the mailing list. Figure 16 shows how reviews are formatted and made ready to be sent to the mailing list.

## 4.7 Linking Patches to Commits

We explained above in Section 4.5.6 how we can create reviews on a particular commit from the messages stored on the mailing list. To make sure that a review created from the message on the mailing list is stored in the correct commit tree on the 'review branch', we had to first determine the mapping between a message on the mailing list and the commit in the project's Git repository.

In the past, Jiang *et al.* [28] performed an evaluation to see which patches uploaded on the Linux kernel mailing list were eventually implemented in Torvalds' main repository. They determined which patches on the mailing list were associated with a commit in Torvalds' Git repository. They performed a checksum matching technique.

Comparing Git and git-reviewed Commands	
git init: Creates an empty Git repository or reinitializes an existing one.	git reviewed - -init: Initializes git-review in order to start reviewing.
git log: Shows commit logs for the repository.	git reviewed - -log: Shows list of reviews made on commits.
git log - -author="torvalds": Shows the commits made by a particular author.	git reviewed - -log-reviewer: Shows the log history of the reviews made by the reviewer.
git show: Shows various types of objects. For blobs, plain content is shown.	git reviewed - -view: Shows the details of the reviews along with the comments made by the reviewers.
git commit - -amend: Amends a commit.	git reviewed - -amend: Amends the last review made.
git format-patch: Prepares a patch for an email submission.	git reviewed - -format: Prepares a review for an email submission.
git push: Pushes local changes to the origin server.	git reviewed - -push: Allows users to push local reviews to the origin server.
git pull: Allows fetching and merging with another repository or a local branch.	git reviewed - -pull: Allows fetching and merging the reviews from the origin.
git rm: Remove files from the working tree.	git reviewed - -rm: Removes the reviews from the 'review' branch.

Table 2: Git commands compared to git-reviewed commands



```

murtuza@murtuza-ThinkPad-X230:~/kernel-git-gregkh-usb$ git review --format 46dd71bf22b2c4f87c6235fb659671f26d64a9be
[REVIEW] [PATCH 3.11 071-233] x86-efi: Fix off-by-one bug in EFI Boot Services reservation.patch (~/.kernel-.../gregkh-usb) - gedit
Open Save Undo
[REVIEW] [PATCH 3.11...es reservation.patch ✕]
1 From : Luis Henriques <luis.henriques@canonical.com>
2 commit: a7f84f03f660d93574ac88835d056c0d6468aebe
3 Message-Id : <1391773652-25214-72-git-send-email-luis.henriques@canonical.com>
4 Date : Fri, 7 Feb 2014 11:44:50 +0000
5 Subject: [PATCH 3.11 071/233] x86/efi: Fix off-by-one bug in EFI Boot Services reservation
6 References: <1391773652-25214-1-git-send-email-luis.henriques@canonical.com>
7 In-Reply-To: <1391773652-25214-1-git-send-email-luis.henriques@canonical.com>
8 Cc: Dave Young <dyoung@redhat.com>, Matt Fleming <matt.fleming@intel.com>, Luis Henriques
  <luis.henriques@canonical.com>
9 3.11.10.4 -stable review patch. If anyone has any objections, please let me know.
10
11 -----
12
13 From: Dave Young <dyoung@redhat.com>
14
15 commit a7f84f03f660d93574ac88835d056c0d6468aebe upstream.
16
17 Current code check boot service region with kernel text region by:
18 start+size >= __pa_symbol(_text)
19 The end of the above region should be start + size - 1 instead.
20
21 I see this problem in ovmf + Fedora 19 grub boot:
22 text start: 1000000 md start: 800000 md size: 800000
23
24 Signed-off-by: Dave Young <dyoung@redhat.com>
25 Acked-by: Borislav Petkov <bp@suse.de>
26 Acked-by: Toshi Kani <toshi.kani@hp.com>
27 Tested-by: Toshi Kani <toshi.kani@hp.com>
28 Signed-off-by: Matt Fleming <matt.fleming@intel.com>
29 Signed-off-by: Luis Henriques <luis.henriques@canonical.com>
30 ---
31 arch/x86/platform/efi/efi.c | 2 +-
32 1 file changed, 1 insertion(+), 1 deletion(-)
33
34 diff --git a/arch/x86/platform/efi/efi.c b/arch/x86/platform/efi/efi.c
35 index 220fa52..f19284d 100644
36 --- a/arch/x86/platform/efi/efi.c
37 +++ b/arch/x86/platform/efi/efi.c
38 @@ -440,7 +440,7 @@ void __init efi_reserve_boot_services(void)
39      * - Not within any part of the kernel
40      * - Not the bios reserved area
41      */
42 -   if ((start+size >= __pa_symbol(_text)
43 +   if ((start + size > __pa_symbol(_text)
44         && start <= __pa_symbol(_end)) ||
45         !e820_all_mapped(start, start+size, E820_RAM) ||
46         memblock_is_region_reserved(start, size)) {
47 --
48 1.8.3.2
49

```

Figure 16: Uploading a review to the mailing list git-reviewed

They merged all the lines which had been changed from the patch, appended it to the file name and calculated the MD5 checksum of that line. After performing the same procedure for Git commits and calculating its relevant checksum, they matched the two to find a link between the patches in the emails and the Git commits.

We used German's implementation based on a similar idea and applied heuristics to do the linking as explained below:

1. We compared each line from the messages in the mbox file with the lines changed in each commit.
2. After getting relevant matches, we computed the most number of matches between the lines changed in each commit and the lines in the messages.
3. The commit which corresponded to the maximum line matches was stored along with the id of the review discussion on the mailing list. This mapping is used by our online website explained in Section 5.2.1.
4. Eventually a review is created out of the messages in the mbox files and committed in the corresponding commit tree on the 'review' branch.

We performed an evaluation on randomly selected messages which consisted of at least a single patch to see how well we were able to link the discussion on the mailing list to a particular commit in the Git repository. The result of the evaluation is given below:

#### **4.7.1 Comparing Linking Results for Linux, Git and PostgreSQL**

We evaluated 30 randomly selected messages on the Linux kernel, Git and the PostgreSQL mailing lists to see if they are matched to a commit. We found that there were correct matches, incorrect matches and no matches. We realized that even a manual search could not find an associated commit as not every commit is reviewed or accepted [28]. We discuss the results in the following subsections:

### **Linux Kernel Mailing List Linking Evaluation**

Our evaluation on 30 randomly selected messages on Linux kernel mailing list gave us good results. We found that our technique linked 26 out of 30 messages to their respective commits with no incorrect matches, while the others remained no matches.

### **Git Mailing List Linking Evaluation**

The messages on Git had similar results when they were linked to a commit. We found that 22 of the messages had correct mapping, 1 was mapped incorrectly and the rest were not mapped.

### **PostgreSQL Mailing List Linking Evaluation**

For the messages on the PostgreSQL mailing list, our technique linked 17 messages correctly to their respective commits, while 2 of them were incorrectly linked and the rest were no matches.

We see that our technique worked best for messages on the Linux kernel mailing List as it linked most of the messages to its respective Git commit on Torvald's repository. A possible reason for our technique to give average results for the PostgreSQL project was that the messages on the PostgreSQL mailing list contain multiple patchsets in a single email thread, thereby resulting in a message to be linked with multiple commits and in such cases the best commit hash cannot be determined.

We invited software developers working on various OSS projects to evaluate our tool. A detailed description of the evaluation and their feedback is given in the next chapter.

## **4.8 Comparing Linux Reviewing Process and git-reviewed Reviewing Process**

We designed our tool in a way that would not change the reviewing ethics of the developers of Linux project as they have been used to it for many years. Below we explain how the reviewing process for git-reviewed is similar to the reviewing process for the Linux project:

Linux Reviewing Process	git-reviewed Reviewing Process
A reviewer subscribes to the mailing list in order to receive notifications for all the email discussions that take place on the mailing list.	A reviewer will install the tool locally and will use git-reviewed commands to pull in all the reviews from the remote repository onto the local machine.
A potential reviewer decides which part of the system or which changes are interesting to him and searches for the review discussions around it using a search engine or other methods.	A potential reviewer will search the review discussions around an interesting commit and view the list of reviews for a particular commit using git-reviewed list commands.
A developer will make the required changes in the local repository and with the help of Git or other methods, broadcast the patch onto the mailing list.	This step is not required as with git-reviewed, discussions travel along with the commits. One can simply make the changes in the repository which can be pulled by other reviewers.
A reviewer receives the patch on the mailing list and performs review.	An interested reviewer will view the review commit by using the git-reviewed command to create reviews which will load a text editor with the contents of the commit. This review gets stored on the review branch.
The reviewer sends a review response in the email thread back to the mailing list.	The review gets created and stored in a threaded manner and the reviewer pushes these reviews onto the remote repository.

Table 3: Comparing Linux reviewing process with git-reviewed reviewing process

## Chapter 5

# Evaluation

Our evaluation of git-reviewed was done by professional developers who have experience with peer review. We did not choose students to evaluate this tool as students are not trained to perform basic reviewing, let alone the style of massively distributed review that is seen on OSS projects as described by Rigby [36]. It was not easy to get developers to try out a tool which they are not familiar with as they are busy with their own work dealing with over 100 reviewing requests on a daily basis [36]. We tried to keep the installation of git-reviewed simple and also made sure that the risk associated with the tool was minimal. We did not alter Git or the developer's data and the tool adds a detached review branch. To remove git-reviewed one simply delete this branch and the associated git-reviewed binary. We created reviews from all the mails on the Linux, Git and PostgreSQL mailing lists from the year 2012 onwards.

We sent out personalized requests via email to the Linux developers to try out the tool and to provide feedback. We also sent out general requests to the entire Linux, Git and PostgreSQL developers' community. Six developers provided their responses. We provide an overview of the feedback we received, what we learnt from the discussions with the developers and how it changed our tool. The positive and negative feedback made by each developer is provided in the subsequent sections.

The evaluation of this tool and the feedback received is discuss in the following sections:

- Distributed Review
- Traceability

## 5.1 Distributed Review

In terms of the responses we received from the developers, we realized that in most cases, developers prefer using a centralized review practice. They are not willing to switch to a distributed style of reviewing. Iwai, a subsystem maintainer for Linux and King, a Git developer, liked how reviews were embedded into the Git repository and provided support for the existing Git tools. Kroah-Hartman, a core developer at Linux, however, preferred creating reviews in a central place where he could discuss problems. On these projects, this place remains the developers' mailing list. He stated:

*“My email client handles reviews just fine, I don't want to have to commit a patch before I can comment on it. Git is at the ”end” of the workflow for subsystem maintainers.”*

While development in the Linux kernel remains a distributed process where developers make changes to the source code, create branches, track back through the version history, however, the reviewing needs to be more centralized. After interviewing developers we realized that they still need a centralized place where discussions take place as opposed to the distributed reviewing methods suggested by us.

## 5.2 Traceability

While reviews need to be done in concert with other developers (i.e. in a centralized manner), the need to lookup which parts of the system have been reviewed, remains important for many developers.

Iwai was positive about the tool, but wanted a better interface for viewing the review discussion:

*“It's nice command line things that are aligned well with the existing git tools. Direct view with github isn't too bad, but a better GUI would be definitely helpful, so that you can surf reviews more easily by pointing a commit id”.*

On the other hand, Kroah-Hartman indicated that the current practice was more than adequate and other techniques for linking were unnecessary. The current practice of linking reviews to a problematic commits is to use a search engine to search for the subject line in the commit log (e.g. “[PATCH] finish\_tmp\_packfile():use strbuf for pathname construction”) has worked well for over 20 years.

On the Git developer mailing list, some developers did not want the entire review discussion included with the commit and just wanted a link to the mailing list discussion (i.e. the message id).

The PostgreSQL developers too just wanted a simple way of viewing discussions around a commit. We realized that in spite of distributed review not being a good fit into the current reviewing processes, traceability still is an important aspect missing for the reviewing techniques. Based on the feedback, we created a simple website where a developer can type in a commit and get redirected to one of the following links to the review discussion of that commit: the message id, a redirect to the archived discussion of the review, or a redirect to the git-reviewed review hosted on GitHub.

### 5.2.1 git-reviewed Tracker

The GitHub interface for viewing reviews linked to particular commits was not user-friendly when it came to handling about 100K reviews for large projects like Linux. In order to make it easier for users of git-reviewed to find the review discussions associated to a particular commit, we created a web page which is shown in Figure 17. This web page is created using PHP, HTML and JavaScript commands which allows a user to enter the commit hash for which he or she wishes to view discussions for. We created a SQLite database which consisted of all the message ids mapped to their respective commit id and would extract the relevant message ids when the user inserted the commit id on the web page. This web page is hosted on <http://cesel.encs.concordia.ca/git-reviewed-tracker.php>.

## Git Reviewed Tracker

Enter Commit Hash To See Review Discussions:

3ee8944fa508675caad9b045af89bc5d845952f3

Show Email Archive

Show Message-Id

Show on GitHub

Figure 17: Track reviews online

This feature may not solve the purpose of being truly distributed as users would need to go to the website in order to find discussions related to a commit, but this definitely solved as a good alternative to pulling in all the reviews locally for a particular repository which are huge in size and require a lot of memory space.

### 5.2.2 Comparison with Trackgit

Rast, a developer for the Git project, created a ‘backward patch tracker’ which tracked the commits to the review discussions on the Git mailing list [5]. This tool allows the storage of an email message-id as a note on each commit. This tool initially used a patch\_id heuristic to match the commits to the email discussions but later added support for the author and date heuristic.

When an evaluation on 30 randomly selected messages was performed on Rast’s commit-message linking technique [5] we found similar accuracy rates as compared to ours as explained in Section 4.7.1. When Rast’s technique was applied to messages on the Linux kernel mailing list, we found that 26 out of 30 messages matched correctly to a commit in the Git repository. For the Git mailing list, 23 messages matched correctly and none of the messages were incorrectly linked. With messages on the PostgreSQL mailing list, we found that 15 messages were linked with their correct commit-id, 4 were not linked correctly and the rest were not matched at all.

Rast mentions the inability of the tool to cope with patch series that are not in the Git repository appearing out of sequence to the mail reader.

He makes use of a Git feature called git-notes in the tool that he created. Git



allows users to add, remove or read notes attached to the various objects without changing the object by making use of the standard command `git-notes` [7]. This makes it easier for the users to add data to any object without actually changing the SHA-1. When a user wishes to add notes to a specific commit, the default editor opens up where the notes are entered. Git also integrates notes with the `git-log` command. This allows users to go through all the notes made on each commit which are automatically appended in the log output.

This newly added feature may work well in adding some comments to a commit without changing the SHA-1 of the commit, but it cannot be used to actually performing distributed peer reviewing in an efficient way. We compared the tool with `git-reviewed` to find out the reasons why `git-notes` would not be a good fit for this kind of implementation:

**Merging git-notes:** Collaboration using `git-notes` is very difficult. If one tries to pull and edit someone else's notes, then pushing these edited notes back to the origin leads to a merge conflict. This has to be done by checking out the notes first and then merge it. This prevents people to use `git-notes` locally without pushing them to the origin and so peer reviewing is not facilitated [10]. With `git-reviewed` there is never an issue with merging. It means that whenever a review is pulled locally and edited, a new review blob object is created. This prevents any merging conflicts and so collaboration using `git-reviewed` becomes easy.

**One Note per Commit:** Git allows only one note per commit. However, it has a way of having multiple name spaces for notes but it may sometimes get confusing for the users. `git-reviewed` on the other hand allows you to have as many comments per review. All reviews for that commit are stored in a single tree and so traceability is high.

**Email Notes to the Mailing List:** There is no way a user can directly send the notes to the mailing list. `git-reviewed` allows you to prepare a review for an email submission. This is an added advantage of `git-reviewed` over `git-notes`.

After receiving an overwhelming response from the developers, we made changes to our tool as per their requirements. We now move on to the next chapter where we provide concluding remarks to our thesis and also discuss avenues for future work.

## Chapter 6

# Conclusion and Future Work

### 6.1 Conclusion

In this work we examined how tool supported review is conducted and provided an overview of how peer reviewing is performed on the Linux kernel mailing list along with the advantages of working in a distributed environment (Chapter 3, 2). Pearce wanted a better way of distribution with his tool Gerrit [43] and Iwai pointed out the lack of good linking and traceability between the commits and review discussions on the Linux kernel mailing list [36]. Keeping these developer requirements in mind, we designed and implemented a lightweight distributed peer review tool git-reviewed, and described the underlying architectural details (Chapter 4).

We linked patches reviewed in the mailing list to the correct commit on the project's Git repository. We found out that in spite of the complex techniques used in the past to link patches to the commits, our simple heuristics resulted in better accuracy rates.

We performed an evaluation of our tool by inviting leading developers of various OSS projects. We gathered their feedback and made some modifications to our tool (Chapter 5). The way git-reviewed was embedded into Git with minimal modifications was liked by developers. We also realized that some of the developers who have been using centralized peer reviewing techniques for many years were reluctant in adopting a new form of reviewing practice overnight, given that distributed peer reviewing is still in its initial stages.

We conjecture that unlike distributed version control systems where developers work independently and then share changes, reviewing needs a centralization point

where discussions can occur. However, traceability is still useful in finding review discussions around a particular commit. Thus, we created an online webpage to allow the developers to look up discussions by entering the commit hash.

We discuss the prospects for future work in the next section.

## 6.2 Future Work

git-reviewed allows users to view the discussions related to a single commit on the mailing lists. However, viewing review discussions around multi-commit branches would add an interesting dimension to this work. We would also like to know how much review has been performed on each subsystem of the project by providing a graphical visualization of the parts of the system that have changed.

Moreover, right now we track reviews from three OSS projects which follow email based code reviews. We intend to pull in reviews from Gerrit which is a centralized code review tool, and try to fit them into the distributed code review environment such as git-reviewed. We wish to maintain reviews for more projects in the near future.

# Bibliography

- [1] 10 million repositories [online].  
<https://github.com/blog/1724-10-million-repositories>.
- [2] Bugzilla [online]. <http://www.bugzilla.org/about/>.
- [3] Bugzilla: Review [online]. <https://wiki.mozilla.org/Bugzilla:Review>.
- [4] Crucible customers [online]. <https://www.atlassian.com/company/customers/customer-list?tab=crucible>.
- [5] Fun things with git-notes, or: patch tracking backwards. <http://git.661346.n2.nabble.com/RFC-RFH-Fun-things-with-git-notes-or-patch-tracking-backwards-td2297330.html>.
- [6] Git documentation. <http://git-scm.com/doc/>.
- [7] git-notes manual page [online]. <https://www.kernel.org/pub/software/scm/git/docs/git-notes.html>.
- [8] How the development process works [online]. <https://www.kernel.org/doc/Documentation/development-process/2.Process>.
- [9] The linux-kernel mailing list faq. <http://www.tux.org/lkml/#s3-7>.
- [10] Note to self [online]. <http://git-scm.com/blog/2010/08/25/notes.html>.
- [11] Postgresql documentation [online]. <http://www.postgresql.org/about/>.
- [12] Rant about github pull-request workflow implementation.
- [13] Review board [online]. <http://www.reviewboard.org/docs/>.

- [14] Rietveld code review, hosted on google app engine [online]. <http://code.google.com/p/rietveld/wiki/CodeReviewHelp>.
- [15] Rietveld users: Who uses rietveld? <https://code.google.com/p/rietveld/wiki/RietveldUsers>.
- [16] Using pull requests [online]. <https://help.github.com/articles/using-pull-requests>.
- [17] Jai Asundi and Rajiv Jayant. Patch review processes in open source software development communities: A comparative case study. In *HICSS: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 10, 2007.
- [18] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering*, 2013.
- [19] Earl T. Barr, Christian Bird, Peter C. Rigby, Abram Hindle, Daniel M. German, and Premkumar Devanbu. Cohesive and isolated development with branches.
- [20] Michael E. Bays. *Software Release Methodology*. 1999.
- [21] C. Bird, A. Gourley, and P. Devanbu. Detecting patch submission and acceptance in oss projects. In *MSR: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 4. IEEE Computer Society, 2007.
- [22] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. The promises and perils of mining git. In *MSR '09: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, page 10, Washington, DC, USA, 2009. IEEE Computer Society.
- [23] L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE: groupware for code inspection. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 169–181. ACM Press, 1990.
- [24] Jason Cohen. *Best Kept Secrets of Peer Code Review*. Smart Bear Inc., 2006.

- [25] E.Schindler. Doing spot-on code reviews with remote teams [online]. <http://www.networkworld.com/news/2008/122308-doing-spot-on-code-reviews-with.html?page=1>.
- [26] Michael Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3):182–211, 1976.
- [27] Michael Fagan. A history of software inspections. *Software pioneers: contributions to software engineering*, Springer-Verlag, Inc., pages 562–573, 2002.
- [28] Yujian Jiang, Bram Adams, and Daniel M. German. 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*, MSR '13, pages 101–110, Piscataway, NJ, USA, 2013. IEEE Press.
- [29] Velusamy Sembugamoorthy Laurence R. Brothers and Bellcore Adam E. Ir-gon. Knowledge-based code inspection with icicle. IAAI-92 Proceedings, AAAI ([www.aaai.org](http://www.aaai.org)).
- [30] M.G.Bradac, D.E.Perry, and L.G.Votta. Prototyping a process experiment. In *Fifteenth International Conference on Software Engineering*, 1993.
- [31] Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar, and Shreya Rawal. The role of patch review in software evolution: an analysis of the mozilla firefox. In *International Workshop on Principles of Software Evolution*, pages 9–18, 2009.
- [32] DE Perry, A. Porter, MW Wade, LG Votta, and J. Perpich. Reducing inspection interval in large-scale software development. *Software Engineering, IEEE Transactions on*, 28(7):695–705, 2002.
- [33] Nicolas Pitre. Re: Figured out how to get mozilla into git. <http://permalink.gmane.org/gmane.comp.version-control.git/21531>.
- [34] Susan Potter. Git, the architecture of open source applications. <http://www.aosabook.org/en/git.html>.
- [35] Julian Ratcliffe. Moving software quality upstream: The positive impact of lightweight peer code review. In *Moving Quality Forward*, 2009.

- [36] Peter C. Rigby. Understanding Open Source Software Peer Review: Review Processes, Parameters and Statistical Models, and Underlying Behaviours and Mechanisms. [thechiselgroup.org/rigby-dissertation.pdf](http://thechiselgroup.org/rigby-dissertation.pdf), Dissertation, 2011.
- [37] Peter C. Rigby, Earl T. Barr, Christian Bird, Premkumar Devanbu, and Daniel M. German. What Effect does Distributed Version Control have on OSS Project Organization. In *Proceedings of the International Workshop on Release Engineering*. IEEE, 2013.
- [38] Peter C. Rigby and Christian Bird. Convergent software peer review practices. In *Proceedings of the the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*, 2013.
- [39] Peter C. Rigby, Brendan Cleary, Frederic Painchaud, Margaret Anne Storey, and Daniel German. Contemporary peer review in action: Lessons from open source development. In *IEEE Software*, pages 56–61. IEEE, Nov-Dec 2012.
- [40] Peter C. Rigby, Daniel M German, Laura Cowen, and Margaret-Anne Storey. Peer Review on Open Source Software Projects: Parameters, Statistical Models, and Theory. *To appear in the ACM Transactions on Software Engineering and Methodology*, page 34, August 2014.
- [41] Peter C. Rigby and Margaret Anne Storey. Understanding broadcast based peer review on open source software projects. In *Proceeding of the 33rd international conference on Software engineering, ICSE '11*, pages 541–550, New York, NY, USA, 2011. ACM.
- [42] Rob. Beginners level course: What is linux?
- [43] Randal Schwartz. Interview with Shawn Pearce, Google Engineer, on FLOSS Weekly. [http://www.youtube.com/watch?v=C3MvAQmHc\\_M](http://www.youtube.com/watch?v=C3MvAQmHc_M).
- [44] Smart Bear Software. Collaborator short demo [online]. <http://www.youtube.com/watch?v=1MBb21DgRYg&feature=youtu.be/>.
- [45] Linus Torvalds. Linus Torvalds on Git . <http://www.youtube.com/watch?v=4XpnKHJAok8>.

- [46] Linus Torvalds. Re: fatal: serious inflate inconsistency. git Mailing List.
- [47] Lawrence G. Votta. Does every inspection need a meeting? *SIGSOFT Softw. Eng. Notes*, 18(5):107–114, 1993.
- [48] Karl E Wieggers. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Information Technology Series. Addison-Wesley, 2001.



# Appendix A

## Email Invites

This appendix contains the email invite message we sent out to OSS developers as well as a sample of a discussion that we had with Iwai. Public discussion that we had with the Git and PostgreSQL developers available at [Git](#) and [PostgreSQL](#) mailing lists, respectively. We had detailed discussions with developers who responded to the message.

### A.1 Recruitment Email

An email was sent out in order to invite participants who would be investing their time in evaluating the git-reviewed tool and providing their suggestions and feedback.

Hi [insert\_developer\_name],

We have integrated peer reviewing on [insert\_mailing\_list] with git to create a distributed peer review tool. We have linked all the reviews you are involved to each commit in [insert\_repository\_name].

You can view the reviews ordered by date on GitHub on [insert\_link].

The GitHub interface is not intended for this purpose, so if you want to view them locally inside your git repository you can install git-review and a 'review' branch will be created.

Please let us know what you think about this tool as it will help us in improving it. If you'd like us to extract the reviews for a different repository, please let us know.

Regards,

Murtuza & Peter

## A.2 Discussion with Takashi Iwai

The following is an email discussion with Linux's subsystem maintainer, Takashi Iwai. This conversation took place after he tried out git-reviewed.

Hi Murtuza,

sorry for my late response, as I've been really too busy to play with your shiny scripts. Now finally I could find minutes to try out. Here are some comments after a short try:

- It's nice command line things that are aligned well with the existing git tools.
- Direct view with github isn't too bad, but a better GUI would be definitely helpful, so that you can surf reviews more easily by pointing a commit id.
- Can this work like git-notes? That is, showing reviews via git log with an option?
- The installation could be a bit improved. It's easy, but the provided script works only for Debian & co.
- A command git-review already exists (for Gerrit), so this name might be confusing for some people.
- I couldn't see any information, though, about how to gather the reviews and put into the repo. Is it a part of project? Majority of patches are floating rather in each subsystem ML, not in LKML, so each tree may need a different setup.

thanks,

Takashi

# Appendix B

## git-reviewed User Manual

### B.1 Installation

git-reviewed runs a bunch of Perl and Ruby scripts. In order to install git-reviewed, you need to clone the repository:

`git@github.com:mmukadam/git-reviewed.git` or

`https://github.com/mmukadam/git-reviewed.git`

Run the following commands:

1. Run `sudo bash install`
2. Clone the repository you wish to review
3. Run `git-reviewed - -init` in order to initialize the tool to be able to review.
4. Start reviewing
5. Use `git-reviewed - -pull` in order to pull all the reviews made in a repository
6. Use `git-reviewed - -push` to push the reviews onto the repository

### B.2 Man Page

**NAME**

git-reviewed- Creates a review on a commit

## SYNOPSIS

git-reviewed [options] [path]

## DESCRIPTION

Allows the user to make a review on a given commit-hash

## OPTIONS

- -init

Initializes git-reviewed in order to start reviewing.

- -re-init

Re-initializes git-reviewed in case of an error.

- -pull

Fetches reviews from remote repositories.

- -push

Updates remote repositories by adding reviews.

- -log

Shows the log history of the reviews made along with the reviewer details.

- -log commit-hash

Shows the log history of the reviews made along with the reviewer details for a particular commit.

- -log-show

Shows the log history of the reviews made along with raw review content.

- -log-show commit-hash

Shows the log history of the reviews made along with raw review content for a particular commit.

- -log-reviewer reviewer-name

Shows the log history of the reviews made by the reviewer.

- -log-limit value

Shows the log history of the reviews made by the author limited to the value specified.

- -list

Lists all the reviews present on the review branch.

- -list commit-hash

Lists all the reviews present on the review branch for the given commit.

- -filter-reviewer reviewer-name

Deletes all the reviews on the review branch except the reviews made by the reviewer.

- -amend

Allows reviewer to make any change to the last review.

- -view review-hash

Shows the contents of the review.

- -format review-hash

Prepares the review for an email submission.

- -respond review-hash

Allows reviewer to respond to any review

- -display review-hash

Displays the interleaved history in a window.

- -rm review-hash

Deletes the review.

- -rm-before date

Deletes the review before the specified date.

- -rm-after date

Deletes the review after the specified date.

## EXAMPLES

\$ git-reviewed commit-hash

... Creating a review object on a particular commit

Creating Review Object

5c1b78028dca1b7424b5d6a0c888fa829236cda2

\$ git-reviewed- -log

...shows the log history of the reviews made

Review b88a0c3b2c8a5924acefdb99fb50bff3a2dfe7ff

Commit Reviewed 02f8efec7e50c925924bbe3a6160de0a82e8b724

Author: Murtuza Mukadam

AuthorDate: Fri Jun 21 23:50:50 2013 -0400

Reviewer: Murtuza Mukadam

ReviewDate: Sat Oct 12 20:27:29 2013 -0400

\$ git-reviewed- -amend

... allows to make a change to the review

Amending Review Object 838a2d22de4d0bb393f2874bc92734a073757fea

\$ git-reviewed- -view review-hash

... shows the raw content of the review object

Commit Reviewed: 02f8efec7e50c925924bbe3a6160de0a82e8b724

Reviewer: Murtuza Mukadam

Review Date: Wed Jun 19 18:40:38 2013 -0400

content

```
$ git-reviewed- -respond review-hash
```

... allows reviewer to respond to a review

```
Creating Response Review Object b3a3becba4b947a381c53a8444ac239b57acd6b0
```

```
$ git-reviewed- -list
```

... lists all the reviews present on the review branch.

```
Total Number of Reviews: 152
```

```
Review: 188aa11a5c2dbeaec05491a73f94ba931248db65
```

```
Commit Reviewed: add7ad68fe46a9a71bbcfb348f75dfa85f758163
```

```
Review: 2f538aa6cdf4d53ffb7846dbcfd5020ce7aad8bf
```

```
Commit Reviewed: add7ad68fe46a9a71bbcfb348f75dfa85f758163
```