International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

Comprehensive Guidelines for Effective Linux Kernel Patch Submission: Enhancing Contribution Quality and Community Interaction

Anish Kumar

Email: *yesanishhere[at]gmail.com*

Abstract: ***This paper presents an extensive guide for submitting patches to the Linux kernel, focusing on best practices that enhance patch acceptance likelihood and foster positive community interactions. The methodology covers patch preparation, recipient identification, email configuration, submission techniques, and communication strategies during the review process. By adhering to these guidelines, contributors can streamline the submission process, improve kernel development efficiency, and increase their impact on this critical open - source project.***

Keywords: Linux kernel, patch submission, git send - email, commit message, subsystem maintainers

1. Introduction

The Linux kernel, a cornerstone of open - source development, relies heavily on contributions from a diverse, global community of developers. The patch submission process is critical to maintaining the kernel's quality, consistency, and continuous improvement. This paper provides a comprehensive guide to best practices for submitting Linux kernel patches, covering all aspects from initial preparation to final acceptance.

1) Patch Preparation

a) Commit Formation

- Squash changes into a single, well - formed commit
- Base changes on a recent stable or release candidate tag from Linus' tree

b) Commit Message Structure

- Format: "subsystem: brief description of change"
- Include a blank line after the summary
- Provide a detailed description of the change
- Use present tense verbs (e. g., "Add feature" not "Added feature")
- Explain the rationale for the change
- Describe any testing performed, including cross - compilation and config option testing

c) Code Style and Documentation

- Adhere strictly to Linux kernel coding style guidelines
- Ensure proper spelling and grammar in all documentation
- Run scripts/checkpatch. pl before submission
- Perform sparse static analysis

d) Signoff

- Include a "Signed - off - by: Your Name your[at]email. com" line

2) Recipient Identification

a) Utilizing scripts/get_maintainer. pl

- Generate patch file:

```bash
git format-patch -1 --stdout > patch.txt
```

- Identify recipients:

```bash
 ./scripts/get_maintainer.pl patch.txt
```

b) Recipient Selection

- Include all listed maintainers and mailing lists
- Err on the side of including more recipients rather than fewer

3) Email Configuration

a) Installing git - email

- Example for Debian/Ubuntu:

```bash
 apt install git-email
```

b) Configuring git send - email

- Set up SMTP settings:

```bash
 git config --global sendemail.smtpencryption tls
 git config --global sendemail.smtpserver smtp.gmail.com
 git config --global sendemail.smtpuser your.email@gmail.com
 git config --global sendemail.smtpserverport 587
```

- For Gmail, use app - specific password:

```bash
git config --global sendemail.smtppass 'your-app-specific-password'
```

4) Patch Submission

a) Using git send - email

- Basic command structure:

```bash
git send-email -1 --to=maintainer@example.com --cc=list@example.com
```

- Use - s flag to automatically add Signed - off - by line
- Review email before sending by typing "e" when prompted

b) Managing Multiple Patches

- Use [PATCH 0/n] for cover letter
- Number patches sequentially: [PATCH 1/n], [PATCH 2/n], etc.
- Ensure patches apply cleanly in order
- Consider using - - cover - letter with git format - patch

5) Patch Discussion and Revision

a) Communication Etiquette

- Use plain text email for all correspondence
- Follow inline reply convention
- Maintain politeness and professionalism
- Be patient with maintainer response times

b) Handling Feedback

- Be open to criticism and willing to revise
- Respond constructively to all feedback

c) Submitting Revisions

- Use [PATCH v2], [PATCH v3], etc. in subject line for revisions
- Summarize changes from previous versions
- Use - - in - reply - to to maintain discussion thread
- Avoid reposting unchanged patches

d) Follow - up

- Send friendly ping emails if no response after 1 - 2 weeks
- Consider reaching out to other maintainers if needed
- Be persistent but patient

2. Conclusion

Adhering to these comprehensive best practices can significantly improve the patch submission process for the Linux kernel. By meticulously following these guidelines, contributors can enhance the quality of their submissions, facilitate smoother interactions with maintainers, and ultimately contribute more effectively to the development of the Linux kernel. This approach not only increases the likelihood of patch acceptance but also fosters a collaborative and efficient development environment within the Linux kernel community.

The Linux kernel's success as an open - source project relies heavily on the quality and consistency of contributions from its global developer base. By adopting these best practices, contributors play a crucial role in maintaining the kernel's high standards and ensuring its continued growth and improvement. As the kernel evolves, so too may these practices, and contributors are encouraged to stay informed about any changes in the submission process.

References

[1] T. Torvalds and D. Diamond, Just for Fun: The Story of an Accidental Revolutionary. New York: HarperCollins, 2001.

[2] Linux kernel coding style. [Online]. Available: https: //www.kernel. org/doc/html/v4.10/process/coding - style. html

[3] Developer's Certificate of Origin 1.1. [Online]. Available: https: //developercertificate. org/

[4] J. Corbet, G. Kroah - Hartman, and A. McPherson, "Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it, " The Linux Foundation, 2016.

[5] G. Kroah - Hartman, "The Linux kernel development model, " in Proc. Ottawa Linux Symposium, 2005, pp.165 - 173.