Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Simplificando o processo de contribuição para o kernel Linux

A Evolução da ferramenta KernelWorkflow

Rubens Gomes Neto

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Paulo Meirelles
Cossupervisor: Rodrigo Siqueira

São Paulo
2022

Resumo

Rubens Gomes Neto. **Simplificando o processo de contribuição para o kernel Linux: *A Evolução da ferramenta KernelWorkflow***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

O kernel Linux é o maior projeto de desenvolvimento colaborativo de software do mundo, mantendo seu ritmo de desenvolvimento devido ao seu modelo de contribuição único. Esse modelo apresenta algumas particularidades que não são comuns em outros projetos, como o envio das contribuições em formato de *patches* por e-mails de texto. O projeto KernelWorkflow (kw) tem a missão de melhorar a experiência de desenvolvimento e contribuição para o kernel Linux. Para isso, ele oferece uma série de funcionalidades para automatizar e simplificar muitos dos processos envolvidos. Assim, o objetivo deste trabalho foi contribuir para a evolução do kw, melhorando e expandindo suas funcionalidades. Nesse sentido, foram apresentadas e discutidas as três contribuições de maior importância e impacto. A primeira foi a refatoração e padronização do sistema de documentação do projeto, melhorando o entendimento e a acessibilidade de suas funções. A segunda contribuição consiste na criação de um banco de dados para substituir o sistema de pastas e arquivos utilizados para armazenar os dados coletados sobre o uso do kw. A migração para um banco de dados permitirá a expansão dos dados coletados, abrindo portas para análises estatísticas sobre o processo de desenvolvimento para o kernel Linux. A terceira contribuição foi a implementação de uma nova funcionalidade para o kw (comando `kw mail`). Essa funcionalidade atua na etapa de envio das contribuições por e-mail para os mantenedores e listas de e-mail do kernel Linux. Para isso, a ferramenta oferece opções que auxiliam na configuração do sistema de e-mail do usuário para que possa enviar o *patch* corretamente. Além disso, a ferramenta permite a automatização do preenchimento dos destinatários corretos baseado nos mantenedores dos arquivos sendo alterados. Todas as contribuições feitas ao kw durante este trabalho ajudaram na evolução do kw, melhorando sua acessibilidade e ampliando suas funcionalidades.

Palavras-chave: GNU/Linux. Kernel Linux. KernelWorkflow. Software Livre. Comunidades de Software. Desenvolvimento Colaborativo de Software.

Abstract

Rubens Gomes Neto. **Simplifying the contribution process to the Linux kernel: *The Evolution of the KernelWorkflow tool***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

The Linux kernel is the largest collaborative software development project in the world. It has managed to keep up with its pace of development thanks to its unique contribution model. This model presents some peculiarities not commonly present in other projects, such as submitting patches via plain text email. The KernelWorkflow (kw) mission is to improve the development experience of contributing to the Linux kernel. For that, kw offers a series of features that automate and simplify many of the involved processes. With this in mind, the objective of this capstone project was to contribute to enhancing kw by improving and expanding its functionalities. In light of this, we present and discuss three significant contributions from our collaboration during the development of this work. The first one is the complete refactor and standardization of kw's documentation, making it easier to understand and use all of its functions. The second contribution pertains to creating a database model to replace the files and folders system currently in use to store kw's usage data. The database system will expand the collected data, which makes it possible to generate statistical analysis on the Linux kernel development process. The third contribution was the implementation of a new feature of kw, the `kw mail` command. This new functionality sends contributions by email to the maintainers and mailing lists of the Linux kernel. For that, the tool offers options that help in the configuration of the user's email system, allowing the *patch* to be sent correctly. In addition, the tool automates the completion of the correct email recipients based on the maintainers of the files being changed. During this work, our contributions helped evolve kw, improving its accessibility and expanding its features.

Keywords: GNU/Linux. Linux Kernel. KernelWorkflow. Free Software. Open Source Software. Software Communities. Collaborative Software Development.

Lista de Abreviaturas

kw	KernelWorkflow
BD	Banco de Dados
ACID	Atomicidade, Consistência, Isolamento, Durabilidade
API	Interface de Programação de Aplicação
CLI	Interface de Linha de Comando
SMTP	Protocolo de Transferência de Correio Simples
IMAP	Protocolo de Acesso a Mensagem da Internet
HTML	Linguagem de Marcação de Hipertexto
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de Figuras

1.1 Caminho de um *patch* (Kroah-Hartman, 2018) 2
1.2 Processo de desenvolvimento para o kernel 3

2.1 Estrutura do código do kw 7

3.1 Estrutura dos diretórios do sistema de armazenamento de dados do kw . 13
3.2 Modelo teórico do banco de dados 15
3.3 Modelo físico do banco de dados 18
3.4 Fluxo de envio de *patches* 28
3.5 Fluxo de execução do `kw mail` 29

Lista de Tabelas

3.1 Comparação entre TinyDB e SQLite 16
3.2 Domínios mais usados pelos desenvolvedores do kernel 25

Lista de Programas

2.1 Estrutura dos componentes do kw 8
3.1 Configurando `git send-email` 21
3.2 Arquivo de configuração do Git 22
3.3 Configuração `git send-email` usando o `kw mail` 22
3.4 Interface de configuração interativa 23
3.5 Interface interativa de seleção de *template* 26
3.6 Comando reduzido de configuração 26
3.7 Arquivo de *template* para o Gmail 26
3.8 Exemplo de usos do *get_maintainer.pl* 28
3.9 Envio de três *patches* para o kernel 30
A.1 Script utilizado para obtenção da lista de domínios usados no kernel . . . 35
B.1 Primeiro *patch* enviado para o kernel Linux usando a ferramenta `kw mail` 37

Sumário

1 Introdução **1**

2 KernelWorkflow **5**

2.1 Modelo de contribuição 5

2.2 Estrutura do código 6

3 Contribuições **9**

3.1 Documentação 9

3.1.1 Contribuição em números 12

3.2 Banco de dados 13

3.2.1 Análise das ferramentas disponíveis 15

3.2.2 Implementação 17

3.2.3 Contribuição em números 19

3.3 kw mail 20

3.3.1 Configuração do send-email 21

3.3.2 Envio dos e-mails 26

3.3.3 Contribuição em números 30

4 Considerações Finais **33**

Apêndices

A Scripts **35**

B Patches **37**

C PRs integrados ao kw **39**

Referências **43**

Capítulo 1

Introdução

O kernel Linux é um projeto de software livre que foi iniciado em 1991 pelo programador Linus Torvalds, que implementa um núcleo para sistemas operacionais. O kernel serve como camada de abstração para a interação entre os programas e o hardware de uma máquina. Ao longo de seu histórico de desenvolvimento, o projeto cresceu ao ponto de que, em sua versão 5.8, em 2020, o código-fonte do projeto continha aproximadamente 28,4 milhões de linhas com quase setenta mil arquivos (Stewart *et al.*, 2020).

O projeto é desenvolvido de maneira distribuída pelo mundo, por desenvolvedores que contribuem ao projeto por diversas razões, alguns trabalham para empresas que mantém drivers para seus dispositivos, alguns são estudantes e outros apenas entusiastas. O projeto atualmente funciona com ciclos de desenvolvimento que se iniciam a partir do lançamento de uma nova versão e costumam durar aproximadamente dois meses e meio. Por exemplo, em lançamentos recentes, entre as versões 5.0 até a 5.12, o projeto recebeu uma média de 217 alterações por dia, o que significa uma média de 9 alterações por hora (Kroah-Hartman, 2022). Essas contribuições vêm de quase 2000 desenvolvedores, sendo dez a quinze por cento destes desenvolvedores que realizaram sua primeira contribuição ao kernel. Essa alta média no fluxo de alterações do projeto só é possível devido à implementação de um modelo de contribuição desenvolvido ao longo dos anos pela comunidade de desenvolvedores.

O kernel está distribuído em diversos repositórios pela internet e sua versão principal, conhecida como *mainline*, segue sob o controle do desenvolvedor original, Linus Torvalds. Linus é a única pessoa que pode incorporar as alterações propostas ao código-fonte do kernel, porém é impossível que uma única pessoa revise e monitore um fluxo de alterações tão alto como o do projeto. Por essa razão, o kernel é dividido em diferentes subsistemas que lidam com alguma parte específica do código, por exemplo, existem os subsistemas de redes, de manejamento de memória, de dispositivos de vídeo, entre outros. Cada subsistema tem um mantenedor, ou grupo de mantenedores, que mantém uma versão própria do kernel e são responsáveis por revisar e às vezes testar as alterações propostas ao seu subsistema. Posteriormente, esses mantenedores enviam uma seleção de alterações para serem mescladas à *mainline* do kernel. A Figura 1.1 apresenta sucintamente o caminho geral que as alterações seguem até a *mainline*, esse caminho faz parte do ciclo de vida do *patch* (Linux kernel community, 2022).

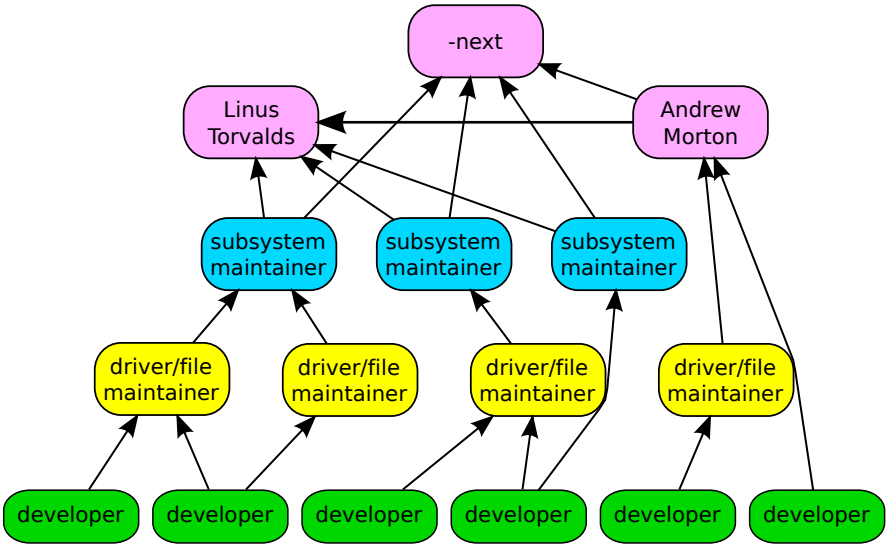


Figura 1.1: *Caminho de um patch (Kroah-Hartman, 2018)*

Cada proposta de alteração é na verdade um arquivo de *patch*, que nada mais é do que um arquivo de texto puro que contém um conjunto lógico de alterações ao código, junto de uma explicação sobre o que foi feito e por quê. *Patches* devem ser enviados para as listas de e-mail de cada subsistema, onde grande parte do processo de desenvolvimento acontece. Para que os desenvolvedores saibam quais são as listas de e-mail relevantes para o subsistema ao qual pretendem contribuir, o kernel possui em seu código-fonte o arquivo *MAINTAINERS*, que agrega as informações sobre os mantenedores de cada subsistema do kernel. Quando esses *patches* chegam nas listas de e-mail, se inicia o processo de revisão do código, onde outros desenvolvedores e mantenedores irão avaliar, testar e propor melhorias para as alterações propostas. A partir disso, o desenvolvedor deve atualizar e re-enviar seus *patches*. Esse processo se repete por quantas vezes e por quanto tempo for necessário para que as alterações estejam estáveis e com o padrão de código esperado no kernel (e nesse momento algum dos mantenedores mescla esse *patch* à árvore do subsistema).

Para que um desenvolvedor envie seus *patches*, é esperado que suas alterações, no mínimo, não quebrem a compilação do kernel. Para isso, um desenvolvedor do kernel costuma seguir um fluxo de trabalho que começa com o planejamento e segue para a implementação das alterações. O desenvolvedor então deve compilar o kernel para garantir que sua alteração não causou problemas. Caso a compilação seja bem sucedida, o próximo passo é instalar essa nova versão do kernel para que possa realizar alguns tipos de validação, que podem ser uma combinação de uma simples inicialização, uma bateria de testes automatizados, e uma série de testes manuais. Por fim, é enviada a primeira versão do *patch* para os mantenedores em questão para revisão. Esse processo está resumido na Figura 1.2.

Dado o grande número de processos e regras a serem seguidas no processo de desenvolvimento, o próprio projeto disponibiliza algumas ferramentas[1] que podem ser utilizadas

[1] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts

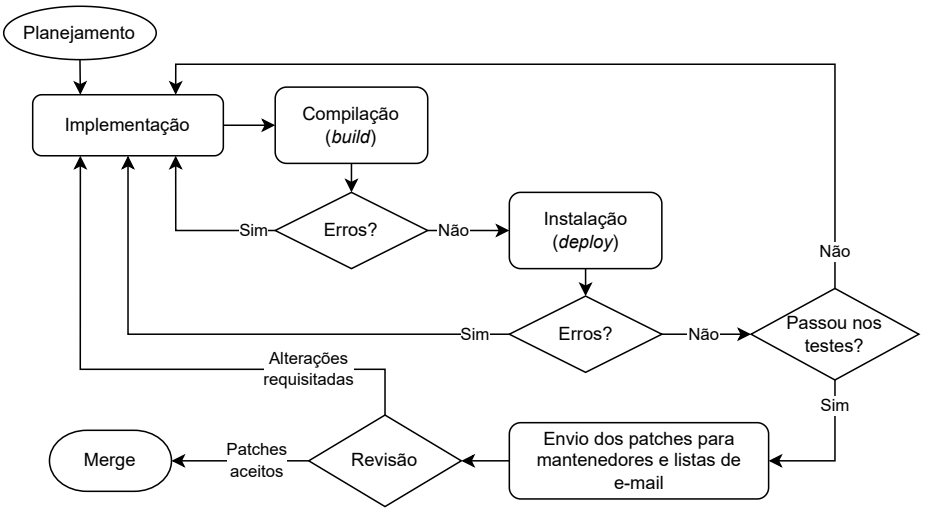


Figura 1.2: *Processo de desenvolvimento para o kernel*

para auxiliar em algumas etapas desse processo. Por exemplo, o *script get_maintainer.pl* pode ser usado para extrair as informações sobre mantenedores e listas de e-mail relevantes de cada arquivo e subsistema disponível. Outro exemplo é o *script checkpatch.pl*, que pode ser usado para garantir que o código do *patch* está seguindo as regras de estilo e formatação impostas pelo kernel. Apesar disso, a disponibilidade dessas ferramentas tem a desvantagem de serem descentralizadas, de forma que os contribuidores precisam aprender e usar muitos *scripts* diferentes. Dessa forma, essas ferramentas não produzem uma experiência unificada para o fluxo de trabalho de desenvolvimento.

Dentro desse contexto de desenvolvimento para o kernel Linux, este trabalho tem como objetivo evoluir a ferramenta KernelWorkflow (kw)[2]. A missão do kw é melhorar a experiência de desenvolvedores do kernel através da automatização e simplificação de muitos dos processos envolvidos. Além disso, o kw busca oferecer uma experiência unificada de desenvolvimento, disponibilizando funcionalidades que atuem em todas as etapas do processo apresentado na Figura 1.2.

Ao decorrer deste trabalho, diversas contribuições foram realizadas para a ferramenta kw com o propósito de evoluir e desenvolver novas funcionalidades da ferramenta. Dentre essas contribuições, três merecem destaque por seu tamanho e impacto, tanto para a ferramenta kw, quanto para a comunidade desenvolvedora do kw e do kernel. O primeiro destaque se dá a refatoração completa da documentação do projeto kw, que possuía um esquema de documentação precário, sendo alterado de maneira a tornar a ferramenta de melhor entendimento, tanto para um iniciante até um experiente mantenedor do kernel. A segunda contribuição de destaque envolve a criação de um modelo de banco de dados para a ferramenta kw, que substitui um sistema interno de manejo de pastas e arquivos. A adoção desse modelo possibilita uma expansão das capacidades de coleta e processamento dos dados que o kw gera. Por fim, a terceira, e mais importante, contribuição que está delineada neste trabalho foi a implementação de uma nova funcionalidade para o kw, o `kw mail`. Essa nova funcionalidade permite que a ferramenta kw atue em uma etapa do processo de

[2] https://kworkflow.org/

contribuição em que ainda não possuía suporte, a etapa de enviar as contribuições para o kernel.

Dessa forma, este texto está estruturado da seguinte forma: o Capítulo 2 é dedicado a apresentação da ferramenta kw, explicitando sua estrutura e também objetivos e propósitos; o Capítulo 3 apresenta e discute as contribuições de destaque, que está separado em três seções dedicadas ao detalhamento de cada uma das contribuições; por último, o Capítulo 4 contém as considerações finais e apontamentos sobre trabalhos futuros relacionados.

Capítulo 2

KernelWorkflow

No contexto do processo de contribuição para o kernel Linux, com suas diferentes etapas e procedimentos, surgiu a ferramenta KernelWorkflow. Inicialmente, pensada como um conjunto de scripts que implementavam automatizações para algumas das etapas do fluxo de trabalho do desenvolvedor, como o processo de compilação e instalação do kernel. Hoje, ela se tornou uma ferramenta que se propõe a oferecer uma experiência unificada e simplificada para desenvolvedores do kernel. Para isso, o projeto vem integrando algumas das ferramentas oferecidas pelo kernel Linux em sua interface, além de outros utilitários que auxiliam no processo de desenvolvimento. Por exemplo, é comum que desenvolvedores do kernel utilizem uma máquina virtual ou remota para que possam testar suas versões do kernel, para isso o kw implementa algumas funcionalidades que permitem que o usuário inicialize a máquina virtual (`kw vm --up`), compile o kernel (`kw build`) e instale o kernel na máquina virtual (`kw deploy`).

A partir da automatização desses processos, e tendo como público-alvo desenvolvedores do kernel Linux, o kw pode presumir alguns comportamentos e com isso oferecer a unificação, e ainda algumas simplificações, das ferramentas disponíveis. Dessa maneira, o kw pode oferecer uma experiência consubstanciada do processo de desenvolvimento e contribuição para o kernel, simplificando o fluxo de trabalho de qualquer contribuidor. Uma outra consequência dessa simplificação é que a ferramenta tem o potencial de diminuir a curva de aprendizado imposta sobre desenvolvedores novatos, podendo, assim, aumentar o número e a retenção de novos contribuidores.

2.1 Modelo de contribuição

Sendo um projeto de Software Livre, o kw segue um modelo de contribuição aberto. O projeto é desenvolvido usando um repositório do sistema controlador de versões Git[1] hospedado no GitHub[2]. Cada repositório pode conter mais de uma versão do programa, conhecidas como *branches*. O repositório do kw possui duas *branches*, a *branch 'master'*

[1] https://git-scm.com/

[2] https://github.com/

que contém a versão estável da ferramenta, e a *branch 'unstable'* que contém a versão mais atualizada possível da ferramenta.

Para realizar contribuições para projetos através do GitHub, primeiro o desenvolvedor deve criar uma cópia do repositório em sua própria conta do GitHub para que tenha permissão de realizar alterações, essa cópia é conhecida como um *fork*. Em sua cópia do repositório o desenvolvedor deve criar uma nova *branch* onde serão feitas as alterações que deseja contribuir. Após realizar as alterações em seu repositório e subir seu trabalho para sua conta no GitHub o desenvolvedor pode abrir um *pull request* (PR) para o repositório do projeto. Cada PR consiste do conjunto de *commits* diferentes entre a *branch* do repositório do projeto em que o desenvolvedor deseja contribuir e a *branch* do repositório do desenvolvedor que contém as alterações sendo propostas.

Após um desenvolvedor criar um PR, esse fica aberto para revisão, onde os mantenedores e membros da comunidade podem ler e comentar sobre a implementação proposta. A partir desses comentários o desenvolvedor deve atualizar seu PR até que esteja de acordo com as expectativas do projeto. Nesse ponto o PR é mesclado ao repositório original e passa a integrar o código-fonte do projeto.

No contexto do kw, todos os PRs devem ser abertos para a *branch 'unstable'*, e todos os *commits* devem conter o nome e e-mail do desenvolvedor. Sempre que relevante o PR deve estar acompanhado de testes e de documentação, e o desenvolvedor deve atualizar seu PR de acordo com as revisões. Quando o PR for aceito ele entra para a *branch 'unstable'*. Quando a versão *'unstable'* está em um estado estável o suficiente o mantenedor do projeto atualiza a *branch 'master'* com a *'unstable'*.

2.2 Estrutura do código

O kw é escrito em Bash[3], seguindo uma série de regras e padrões para manter seu código-fonte organizado e fácil de ser mantido. Alguns desses padrões são impostos por ferramentas externas como o *shellcheck*[4], que ajuda a evitar alguns erros comuns em Bash, e o *shfmt*[5], que impõe um padrão e estilo ao código. Outras regras utilizadas estão descritas na seção de *code-style* da documentação do projeto[6]. Para além dessas regras, o código do kw está organizado como ilustrado na Figura 2.1, separado em cinco partes: um ponto central de interação (arquivo *'kw'*), componentes, bibliotecas, plugins e documentação. A partir disso, podemos fazer uma análise mais profunda sobre cada uma dessas partes.

Ponto central de interação (arquivo *kw*). O kw possui várias funcionalidades que realizam operações que não estão diretamente relacionadas entre si. Por exemplo, o componente de `deploy` não é diretamente relacionado com o componente `configm`, que gerencia os arquivos *.config* do usuário. Para isso, o kw utiliza o arquivo *'kw'*, que funciona como ponto central de interação do usuário com todos os componentes do kw. Em outras palavras,

[3] https://www.gnu.org/software/bash/

[4] https://www.shellcheck.net/

[5] https://github.com/mvdan/sh

[6] https://github.com/git/git/blob/master/Documentation/CodingGuidelines

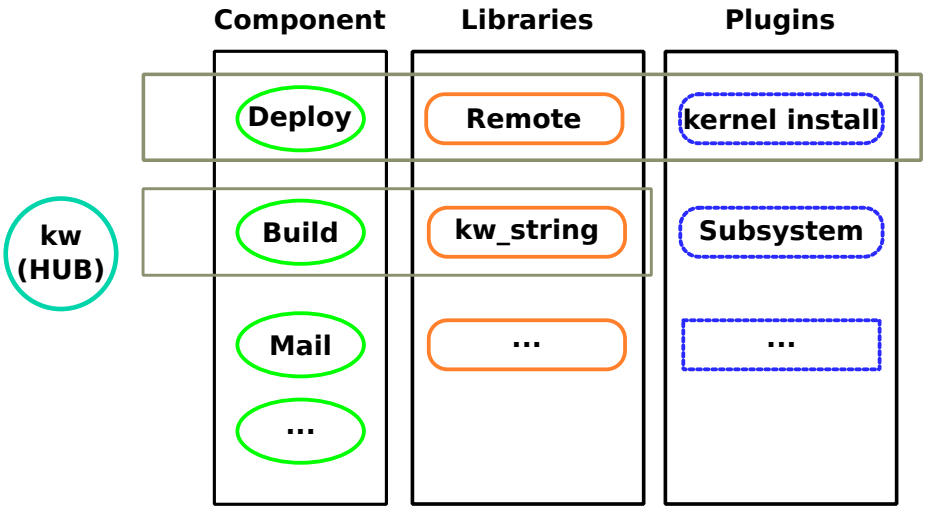


Figura 2.1: *Estrutura do código do kw*

todos os comandos do kw passam primeiro pelo arquivo *'kw'*, que por sua vez é responsável por invocar os componentes necessários para a execução de cada comando.

Componentes. Cada funcionalidade do kw é tratada como um componente específico, geralmente com um arquivo dedicado. Por exemplo, a implementação da funcionalidade de `deploy` pode ser encontrada no arquivo *'src/deploy.sh'* e a implementação da funcionalidade de `build` pode ser encontrada no arquivo *'src/build.sh'*. A maior parte dos arquivos de componente apresentam a estrutura básica apresentada no Programa 2.1. Essa estrutura padronizada facilita a leitura e o entendimento dos programas, pois a função principal do componente, que gerencia todo o funcionamento do mesmo, aparece logo no começo do arquivo e pode ser usado para guiar a leitura do arquivo. Além disso, as funções que lidam com o tratamento das opções e de ajuda ficam posicionadas ao final do arquivo, deixando o espaço do meio dedicado à implementação de fato da funcionalidade.

Bibliotecas. Como o nome sugere, estes arquivos funcionam como bibliotecas dentro do kw. Eles contém implementações de funções e utilidades que são compartilhadas entre várias das funcionalidades. Por exemplo, existem arquivos que lidam com a manipulação de strings e com acesso remoto.

Plugins. Para manter a coesão, o kw busca manter isolados na pasta *'plugins'* programas que contém implementações que são suscetíveis a alterações externas, altamente mutáveis ou com aplicações muito específicas. Por exemplo, o kw possui um plugin que lida com o *Direct Rendering Manager* (DRM), especificamente para a comunidade de desenvolvimento de GPU do kernel.

Documentação. O kw mantém um diretório com arquivos dedicados para a documentação do projeto. Eles são usados para gerar as páginas de manual e a versão em HTML que

Programa 2.1 Estrutura dos componentes do kw

```
1   # função de entrada para a funcionalidade
2   # arquivo 'kw' chama essa função
3   function <componente>_main()
4   {
5       ...
6   }
7   ...

    # Implementação do componente

138 ...
139 # essa função lê e interpreta os argumentos passados
140 function parse_<componente>_options()
141 {
142     ...
143 }
144
145 # função que imprime a ajuda curta, ou abre as páginas de manual
146 function <componente>_help()
147 {
148     ...
149 }
```

é usada no site[7]. A compilação da documentação é feita através da ferramenta Sphinx[8], que é a mesma utilizada pelo kernel Linux.

Durante o desenvolvimento deste trabalho, foram feitas contribuições em todas as partes do projeto, tendo três contribuições de maior escopo e impacto. A primeira dessas contribuições foi a refatoração do sistema de *documentação* do projeto, trazendo melhorias e padronizações. A segunda contribuição de destaque é a criação de um modelo de banco de dados que envolveu a implementação de uma nova *biblioteca* e alterou o funcionamento de alguns dos arquivos. Por fim, a criação de um novo *componente* chamado `kw mail` que possibilita o envio de *patches* por e-mail para os mantenedores e listas de e-mail do kernel Linux. Como requerido pelos guias de contribuição e estilo de código do projeto, todas as contribuições foram acompanhadas de testes de unidade para garantir seu funcionamento, e também uma boa documentação sobre o funcionamento de novas funções e componentes.

[7] https://kworkflow.org/

[8] https://www.sphinx-doc.org/

Capítulo 3

Contribuições

Durante o decorrer deste trabalho diversas contribuições foram feitas ao projeto KernelWorkflow que elevaram o nível da ferramenta e culminaram no desenvolvimento de uma nova funcionalidade que ajuda na missão do kw de oferecer uma experiência unificada e simplificada de desenvolvimento para o kernel Linux. Neste capítulo, serão discutidos alguns detalhes de três contribuições que tiveram maior impacto na ferramenta.

Inicialmente, está detalhada a refatoração da documentação do programa, que padronizou e organizou a documentação do projeto de maneira a melhorar o entendimento e a acessibilidade da ferramenta, promovendo a sua adoção por futuros usuários.

Em seguida, é feita uma discussão sobre a implementação de um modelo de banco de dados junto de uma interface de acesso a esse banco de dados, que deve ser implementada em *bash* para se adequar ao resto do projeto. O intuito é que o novo sistema de dados seja mais robusto do que o sistema de pastas e arquivos atualmente em uso pelo projeto, e possa oferecer melhores opções de estatísticas sobre esses dados.

Por fim, são descritos os detalhes da implementação da nova funcionalidade `kw mail`, que permite que usuários configurem facilmente seus ambientes para que possam realizar o envio correto de contribuições para o kernel. Além disso, a nova ferramenta ainda permite o envio das contribuições para os mantenedores do kernel de maneira mais simplificada e menos confusa do que manualmente.

3.1 Documentação

Todo projeto de *software* que tem alguma pretensão de ser utilizado e adotado por outros depende, além de funcionalidades que atendam as demandas de seus usuários, uma boa documentação que explique seu uso, suas funcionalidades e limitações. Esse fato é ainda mais acentuado em projetos de Software Livre, que possuem um modelo de desenvolvimento distribuído, em que os usuários podem também ter acesso ao código-fonte e vir a contribuir com o projeto[1]. Por essas razões, a documentação de um projeto desse tipo tem também a necessidade de explicar como funciona e como é implementada cada

[1] https://www.gnu.org/philosophy/free-sw.html

uma das funcionalidades oferecidas pelo projeto. Com isso em mente, podemos definir a documentação de um projeto como o conjunto de documentos que contém todos os textos explicativos de uso e funcionamento da ferramenta, mas também todo o conjunto de comentários presentes no código-fonte do projeto, esse último só é acessível aos usuários se o projeto tiver o código aberto.

Podemos observar duas categorias distintas de documentação: aquela voltada para o usuário final e aquela destinada aos desenvolvedores. A documentação voltada para o usuário do *software* costuma ter uma abordagem mais pragmática e focada em explicar como e porque usar a ferramenta, e portanto costuma tomar a forma de guias ou *wikis*. Em contrapartida, a documentação voltada para o desenvolvedor costuma se preocupar mais em explicar como as funções funcionam internamente e muitas vezes o porquê dessa funcionalidade ter sido implementada desta forma.

Um usuário comum dificilmente irá investir seu tempo para ler a documentação destinada aos desenvolvedores, já que muitas vezes essa documentação está "escondida" dentro do código-fonte do projeto. O inverso já não é o mesmo, visto que pode ser importante para o desenvolvedor ter alguma familiaridade com a ferramenta, para que possa também entender melhor os usos de caso da ferramenta.

Uma maneira comum de acessar a documentação de um programa, em particular em programas do estilo *command-line interface* (CLI), é através da criação de *man-pages*[2]. As *man-pages* são textos que descrevem e explicam as funcionalidades e opções do programa a que se referem, e podem ser acessadas através do terminal usando ferramentas como `man`[3]. Algumas vezes também é possível acessar esses textos de documentação em forma de site, de forma a possibilitar o acesso a documentação mesmo sem precisar instalar o programa.

Para que seja mantido um certo nível de coerência na documentação muitos projetos adotam alguma política de padronização da documentação. Dessa forma, é possível garantir que qualquer nova funcionalidade que seja introduzida tenha uma documentação coerente com as de funcionalidades já existentes. No caso de um projeto novo, que está começando a expandir sua documentação e procura implementar uma padronização, pode ser interessante tentar aderir e se adaptar a guias e estilos utilizados por outros projetos já conhecidos e amplamente difundidos. Com isso, as curvas de aprendizado para escrita e entendimento da documentação serão reduzidas pelo fato do modelo adotado já ser familiar aos usuários e desenvolvedores.

No início deste trabalho, a documentação do kw ainda não estava bem desenvolvida, continha apenas algumas explicações básicas sobre suas funcionalidades e essas informações não estavam muito bem organizadas, dificultando o acesso e a compreensão das funcionalidades oferecidas. Além das páginas com guias sobre *'como instalar o kw'*, *'como contribuir para o kw'*, *'como funcionam os testes'*, e *'estilo de código do kw'*, o projeto possuía apenas uma página de documentação no estilo manual. Essa página era extremamente longa, pois continha nela todas as explicações disponíveis para todas as funcionalidades do kw, ou seja, caso quisesse ler sobre o funcionamento da função que lida com o gerenci-

[2] https://www.kernel.org/doc/man-pages/

[3] https://man7.org/linux/man-pages/man1/man.1.html

amento de arquivos *.config* seria necessário acessar este arquivo geral e buscar pela função relevante. Além disso, não havia um padrão sobre o acesso aos textos de ajuda, algumas funcionalidades aceitavam opções como `-h` ou `--help`, e outras não.

A ferramenta kw possuía através de sua CLI uma opção (`kw help`) que imprimia na tela para o usuário um texto de ajuda enxuto, focado em apresentar as opções disponíveis com uma breve descrição. Possuía também a opção `kw man` que abria o arquivo geral já mencionado, que continha explicações mais detalhadas sobre cada uma das funcionalidades disponíveis utilizando o programa *man*. Para melhorar a acessibilidade da documentação do projeto, o comportamento apresentado pelos comandos `kw help` e `kw man` poderiam ser replicados para cada uma das funcionalidades, criando páginas de manual específicas para cada uma.

Como o kw ainda não impunha um padrão de formatação em sua documentação, essa foi a oportunidade para adotar um novo padrão, uma vez que toda a documentação seria revista. Para simplificar esse processo, decidiu-se buscar inspiração na documentação do projeto Git (*CodingGuidelines* 2021), por ser um projeto muito bem consolidado e difundido, especialmente na comunidade desenvolvedora do kernel, já que é o sistema de versionamento utilizado pelo projeto. Além disso, a documentação do projeto possui uma seção que detalha as convenções adotadas na documentação do projeto, dessa forma foi possível usar essas convenções como inspiração para a padronização do kw.

O primeiro passo para que seja possível replicar esses comportamentos nos níveis individuais das funcionalidades oferecidas é a criação de arquivos separados que contenham apenas a documentação relevante o respectivo comando. A documentação de cada funcionalidade do kw pode ser extraída do arquivo manual principal, e as informações foram adaptadas aos novos padrões e quando necessário atualizadas e expandidas. Com isso, cada componente do kw recebeu uma função de ajuda, que imprime na tela uma curta lista dos possíveis usos de cada funcionalidade com breves descrições, de maneira similar ao funcionamento do comando `kw help`.

Alguns dos padrões adotados nesses arquivos são: o uso de colchetes '`[ ]`' para delimitar um parâmetro ou argumento e o uso dos símbolos de menor e maior que '`< >`' para indicar valores que devem ser substituídos pelo usuário, como seu nome ou e-mail por exemplo. Além disso, as páginas de manual foram adaptadas para seguir o modelo das *man-pages*, separando o arquivo nas seguintes seções:

- **NAME**: o nome do comando e uma frase explicativa
- **SYNOPSIS**: demonstra como utilizar o comando e seus argumentos
- **DESCRIPTION**: contém uma descrição do comando e seus propósitos e usos
- **OPTIONS**: lista com todos os argumentos e parâmetros com descrições individuais e específicas
- **EXAMPLES**: exemplos práticos de uso do comando

O próximo passo é possibilitar o acesso a cada uma das novas páginas de manual, que agora são específicas para cada componente. Até então, o kw utilizava o programa

`rst2man`[4] para gerar e acessar a página de manual via terminal, porém utilizava o programa `sphinx`[5] para gerar, a partir dos mesmos arquivos, as páginas de documentação em formato HTML utilizadas no site. Para garantir uma boa integração entre as novas páginas de manual, foram criadas referências conectando esses arquivos, porém essas referências não eram compatíveis com a ferramenta `rst2man`, resultando em problemas na criação das páginas de manual. Para superar esse problema, foi preciso utilizar o `sphinx` para gerar os arquivos no formato de manual, além das páginas em HTML. Dessa forma, todas as diretrizes e parâmetros utilizados seriam compatíveis para os dois formatos.

Aliado a isso foi possível implementar uma otimização na maneira como o kw lida com as páginas de manual. Inicialmente, o kw gerava a página de manual apenas quando o usuário invocava o comando `kw man`, ou seja, a cada uso do comando o kw gerava o arquivo em formato de manual para depois mostrá-lo para o usuário. Ao alterar o programa usado para gerar as páginas de manual, o comportamento de geração dos arquivos pode ser movido para o processo de instalação do kw. Dessa forma, o kw gera todos os arquivos de manual durante a instalação e, quando o usuário desejar ler alguma das páginas, o programa apenas acessa os arquivos do disco.

A interface finalizada da documentação permite que o usuário acesse a lista de parâmetros e opções disponíveis para cada comando utilizando o comando:

```
kw <comando> -h
```

As páginas de manual específicas de cada um de seus comandos podem ser acessadas utilizando qualquer umas das variações:

```
kw man <comando>
kw <comando> --help
```

As páginas geradas possuem uma formatação consistente entre si, que se estende também para a versão HTML das páginas. A página de manual original pode ser reduzida, de maneira que os antigos textos deram lugar para links para as páginas relevantes, deixando no arquivo apenas breves apresentações sobre as diferentes funcionalidades do kw.

3.1.1 Contribuição em números

A nova documentação do kw proporciona um caráter mais profissional ao projeto e, principalmente, garante uma facilidade ao acesso dos usuários aos textos descritivos que podem auxiliar no uso da ferramenta. Tudo isso ajuda a tornar o processo de familiarização com o uso do kw e suas funcionalidades mais direto e descomplicado.

A implementação dessa refatoração da documentação foi feita, em sua maior parte, em um único PR[6], mas que continha mais de 18 *commits*. Um *commit* inicial para implementação do novo sistema de documentação e os outros para a adequação individual de cada componente do projeto. Ao todo foram mais de 42 arquivos afetados pelas mudanças,

[4] https://manpages.debian.org/testing/docutils-common/rst2man.1.en.html

[5] https://www.sphinx-doc.org/en/master/

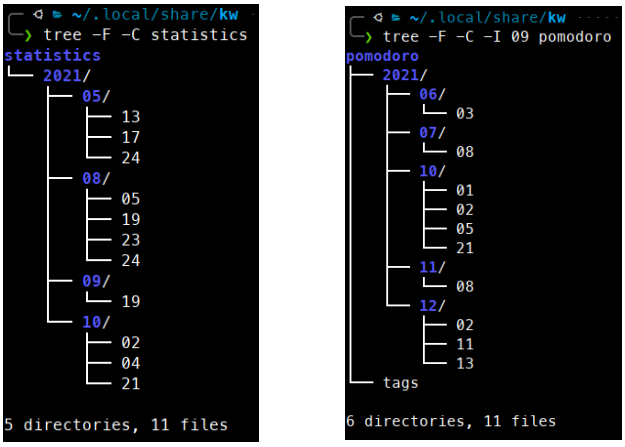
[6] https://github.com/kworkflow/kworkflow/pull/348

totalizando 1112 linhas adicionadas e 470 linhas removidas. O processo de revisão durou aproximadamente 45 dias e o PR recebeu mais de 25 comentários durante o período de revisão.

3.2 Banco de dados

Como parte do funcionamento interno do kw, existe a coleta de alguns dados referentes ao uso de algumas de suas funções, como o tempo de compilação e instalação do kernel. O kw armazena também as informações sobre as sessões de foco iniciadas pelo usuário através do comando `kw pomodoro`[7], que tem seu nome baseado na técnica *Pomodoro* (Cirillo, 2009). Atualmente, esses dados são coletados e armazenados localmente e são usados apenas para gerar algumas estatísticas que podem ser de interesse do usuário. Apesar disso, como o kw possui diversas utilidades voltadas para desenvolvedores do kernel, a ferramenta pode ser usada para gerar dados sobre o processo de desenvolvimento para o kernel Linux. Esses dados têm o potencial de abrir portas para novas pesquisas na área de desenvolvimento e engenharia de software, em especial Software Livre. Para que isso se concretize, porém, é necessário uma reestruturação do sistema de coleta de dados do kw para torná-lo mais robusto e escalável, tanto para suportar uma quantidade de dados maior, como permitir a inclusão de novas métricas.

O modelo de dados empregado pelo kw até então consiste de pastas para cada tipo de dado coletado, mais especificamente, possui uma pasta para armazenar as informações sobre as sessões de *Pomodoro* e outra para os dados referentes ao uso das funcionalidades dos comandos `kw build` e `kw deploy`. Esses dados são utilizados para geração de relatórios de uso de tempo para o caso do `kw pomodoro` e para a geração de estatísticas sobre o uso das ferramentas do kw. Dentro dessas pastas, os dados estão organizados de acordo com a estrutura mostrada na Figura 3.1.



(a) *Statistics* **(b)** *Pomodoro*

Figura 3.1: *Estrutura dos diretórios do sistema de armazenamento de dados do kw*

Conforme ilustrado na Figura 3.1, cada ano é representado por uma pasta, dentro dela existem pastas referentes aos meses, que contém arquivos referentes a cada dia em que

[7] https://kworkflow.org/tutorials/pomodoro-report.html

o kw foi utilizado. O conteúdo desses arquivos contém os dados sobre qual comando foi executado e quanto tempo sua execução levou.

Apesar de funcional, esse modelo apresenta algumas desvantagens. O modelo atual representa um conjunto de dados descentralizado que depende da manipulação de arquivos personalizados no sistema do usuário e, portanto, existe a necessidade do kw manter e implementar funções específicas que lidem com esses arquivos dentro do projeto. Aliado a isso, as informações contidas nos arquivos referentes a cada dia não seguem o mesmo padrão entre as entradas das sessões de *Pomodoro* e das estatísticas dos comandos de compilação e instalação do kernel.

Para melhorar a maneira como o kw lida com os dados que coleta, é necessário que os dados sejam padronizados e reestruturados, além disso seria muito útil se parte do processamento e armazenamento desses dados fosse externalizado do kw. Nesse contexto, uma das soluções mais utilizadas para o gerenciamento de dados é um sistema de banco de dados. Com um banco de dados dedicado, o projeto passa a ter acesso a seus dados de forma centralizada e escalável, visto que muitos bancos de dados implementam otimizações para o armazenamento e acesso a esses dados. Com isso, o kw não precisa implementar seu próprio sistema para manejar pastas e arquivos espalhados pelo sistema do usuário. Além disso, a adoção de um sistema de banco de dados permite que o kw gere análises estatísticas mais sofisticadas sem a necessidade de implementar um sistema próprio para realizar os cálculos, já que existem sistemas de banco de dados que já oferecem ferramentas para realizar esses cálculos.

Para realizar a transição do sistema de dados do kw para usar um sistema de banco de dados é preciso primeiro criar um modelo de banco de dados que comporte todos os dados que o kw já coleta e que possa ser expandido futuramente. Como já mencionado, o kw coleta dados sobre o uso de três de seus componentes, `kw build`, `kw deploy` e `kw pomodoro`. Os dados referentes aos componentes *build* e *deploy* contém as seguintes informações:

- Data de utilização do comando
- Um identificador sobre qual funcionalidade do comando foi utilizada
  - build, deploy, list, uninstall, build_failure, modules_deploy
- Tempo de execução do comando

Os dados coletados do comando `kw pomodoro` são um pouco diferentes, já que o que está sendo monitorado são as sessões do método *Pomodoro* e não sua execução em si. Esses dados são armazenados para que possam ser gerados relatórios de uso através do comando `kw report`. Dessa forma, os dados coletados sobre o `kw pomodoro` são:

- Tags criadas pelo usuário para agrupamento das sessões
- Tag da sessão
- Data de início da sessão
- Hora de início da sessão
- Duração da sessão

- Descrição da sessão

A partir desses dados, é possível criar um modelo para o banco de dados, então, em um esforço de unificar os diferentes dados coletados foi criada a abstração do uso do kw como um evento que ocorreu em uma determinada data e horário, esses eventos podem ser iniciar uma sessão de *Pomodoro*, ou executar um dos comandos do kw. A Figura 3.2 mostra essa primeira abstração em forma de um modelo teórico para o banco de dados.

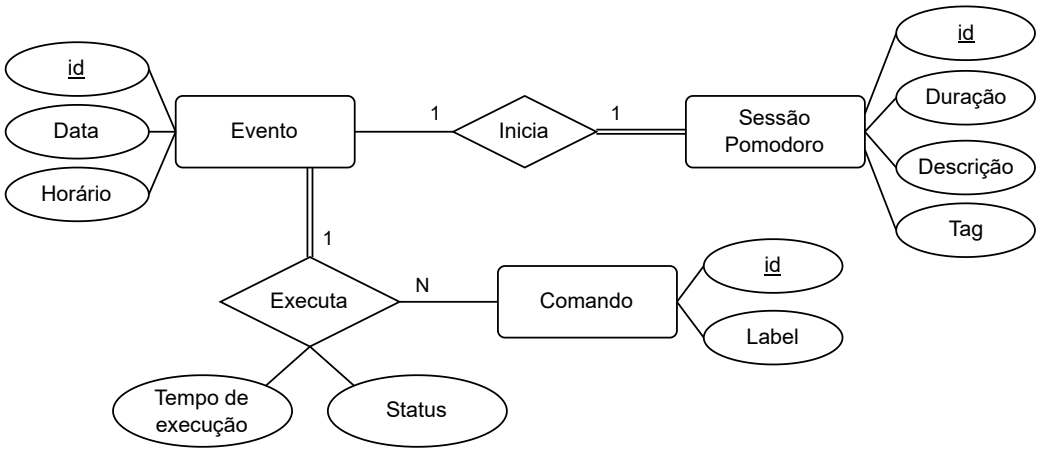


Figura 3.2: *Modelo teórico do banco de dados*

3.2.1 Análise das ferramentas disponíveis

Como o kw é um projeto relativamente pequeno que armazena poucos dados e em um contexto individual, ou seja, os dados manejados pelo kw se referem apenas aos dados gerados pelo usuário, não é necessário um sistema de banco de dados muito complexo. Inclusive, é interessante evitar o uso de dependências muito grandes e manter o kw o mais leve possível. Além disso, seria interessante ter algumas garantias sobre os dados. Para bancos de dados, um conjunto de garantias úteis são de *Atomicidade, Consistência, Isolamento e Durabilidade* (ACID) (Haerder e Reuter, 1983) para as transações sobre os dados do banco. Assim, podemos resumir alguns dos requisitos que guiaram a escolha do sistema de banco de dados a ser usado como:

- Comportar o conjunto de dados
- Poucas dependências
- Ocupe pouco espaço
- Garantias ACID
- Possibilitar a integração com Bash
- Disponível em distribuições de Software Livre, como Debian e Arch

Dessa forma, duas opções foram propostas como candidatas pelo mantenedor do kw,

o *SQLite*[8] e *TinyDB*[9]. Ambas as ferramentas possuem implementações com código-fonte aberto (uma em domínio público e outra sob uma licença de Software Livre, respectivamente) e são voltadas para aplicações leves. Uma comparação entre elas é apresentada na Tabela 3.1.

Requisitos	TinyDB	SQLite
Comportar o conjunto de dados	Sim	Sim
Poucas dependências	Python 3.6+	Biblioteca: - libc6* CLI: - libreadline8 - zlib1g*
Ocupe pouco espaço	180 kB	Biblioteca 1000 kB CLI 3000 kB
Garantias ACID	Não possui	Todas transações ACID
Possibilitar a integração com Bash	Interface apenas em Python	Interface em Bash
Disponível em distribuições de Software Livre	Disponível via `pip`	Disponível via `apt` e `pac`
Licença	MIT	Domínio público

Tabela 3.1: *Comparação entre TinyDB e SQLite*

O *TinyDB* é um sistema extremamente simples e pequeno: o código-fonte possui apenas 1800 linhas de código. Ele é orientado a arquivos, de maneira que o banco de dados fica contido em apenas um arquivo e implementa uma API simples com foco em usabilidade. Essa aplicação é implementada em python e comporta-se como uma biblioteca que gerencia o acesso e a manipulação dos dados dentro de scripts também em python, o que não favorece a adoção pelo kw que é escrito em Bash. Outro ponto a se considerar é que, devido a sua simplicidade, faltam ao *TinyDB* algumas funcionalidades costumeiras de sistemas de gerenciamento de bancos de dados, como o manejo dos relacionamentos entre entidades, a criação de índices e não tem suporte para estruturas de dados mais complexas. Além disso, o banco de dados não oferece garantias de ACID aos comandos, o que deixa o banco de dados mais suscetível a falhas.

Já o motor de banco de dados *SQLite*, apresenta uma aplicação muito mais completa e robusta para a criação e manipulação de bancos de dados. A biblioteca apresenta as garantias ACID, que garantem a integridade do banco de dados mesmo em casos de perda de energia ou falhas no sistema. Além disso, o banco de dados gerado pelo *SQLite* é armazenado em um único arquivo próprio que pode ser transferido e utilizado em máquinas com múltiplas arquiteturas e sistemas operacionais. Outra vantagem de se usar o *SQLite* para o kw, é que o *SQLite* possui uma CLI que pode ser invocada através do shell. Dessa forma, podemos manter a base de código do kw em Bash, adicionando apenas arquivos SQL que contenham a estrutura do banco de dados a ser usado. Além disso, como SQL já é uma linguagem bem estabelecida e especializada para a manipulação de bancos de dados, essa implementação garante uma acessibilidade maior para contribuidores futuros. Dessa forma, apesar da biblioteca *SQLite* ocupar maior espaço em disco, sua maior gama de funcionalidades e garantias a tornam a melhor escolha dentre as opções analisadas.

[8] https://www.sqlite.org/

[9] https://tinydb.readthedocs.io/

3.2.2 Implementação

Interface

Com a escolha do banco de dados a ser utilizado para essa transição, pôde ser implementada no kw uma pequena interface com funções que permitem que os componentes internos possam manipular o banco de dados. Essa interface generalizada foi um bom ponto de partida e pode ser usada como validação da integração do SQLite com o kw. A princípio, os dados coletados pelo kw não precisam ser alterados ou atualizados, por essa razão a interface inicial concentra-se em oferecer suporte para a inserção e leitura dos dados armazenados. Para isso, as funções funcionam como *wrappers* para a CLI disponível para SQLite, `sqlite3`, que abstraem a construção de comandos SQL a partir dos argumentos.

Complementarmente a isso, foram implementadas funções para a execução de comandos SQL arbitrários, bem como a execução de scripts SQL. Dessa forma, se garante uma flexibilidade para a implementação de funções que manipulam os dados de maneira muito específica à funcionalidade e que portanto não fariam sentido como parte da interface geral. Além disso, a possibilidade de executar scripts SQL é essencial e importante, pois permite que a estrutura do banco de dados seja armazenada em um script SQL que pode ser utilizado para criar e atualizar o banco sempre que necessário.

Modelo

O SQLite dá suporte a muitas das funcionalidades da linguagem SQL, porém algumas das implementações diferem da linguagem padrão. Uma das diferenças que merece ser destacada é o uso de estruturas de dados próprias e dinâmicas, que são chamadas de classes de armazenamento por serem mais genéricas do que estruturas de dados convencionais. No caso, o SQLite reconhece cinco classes de armazenamento diferentes, são elas:

- **NULL**: Valor vazio
- **INTEGER**: Número inteiro com sinal, ocupa 0, 1, 2, 3, 4, 6 ou 8 bytes de espaço dependendo de sua magnitude
- **REAL**: Número em ponto flutuante de 8 bytes
- **TEXT**: Sequência de caracteres de texto
- **BLOB**: Sequência de bytes arbitrária

A biblioteca do SQLite reconhece a maior parte dos tipos de dados comumente usados por sistemas gerenciadores de bancos de dados SQL e automaticamente efetua a conversão dos valores para alguma das classes disponíveis. Por exemplo, é comum que valores para datas e horários sejam armazenados em colunas do tipo DATETIME, porém no caso do SQLite esses valores são armazenados como TEXT e o SQLite providencia algumas funções que auxiliam na manipulação e conversão dessas datas para os formatos desejados. Utilizando-se desses conceitos, se torna possível adaptar o modelo teórico para um modelo físico.

Para realizar essa adaptação foram criadas três tabelas principais: a tabela **“event”**, que

guarda as informações de data e hora que um *evento* foi acionado; a tabela **“timebox”**, que armazena as informações da sessão *Pomodoro*, como duração, descrição e *tag* associados; e a tabela **“executed”**, que detém as informações referente à execução dos diversos comandos do kw. Além disso, as *tags* criadas pelo usuário e os identificadores de qual comando foi executado foram separadas em tabelas específicas. Por fim, foi separada em uma tabela o status de execução do comando para evitar o uso de identificadores como o *build_failure*.

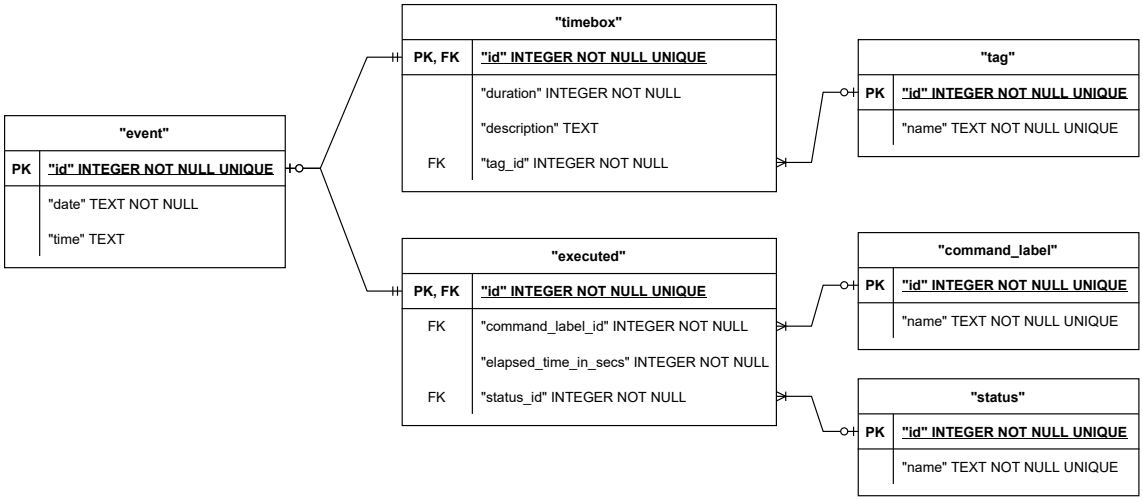


Figura 3.3: *Modelo físico do banco de dados*

A partir desse modelo, podemos destacar, então, a decisão de se utilizar o campo de identificação (*“id”*) das tabelas **“timebox”** e **“executed”** como chave estrangeira que relaciona cada uma delas com a *“id”* da tabela de eventos (**“event”**). Isso mantém as entradas únicas, já que todas as entradas das tabelas **“timebox”** e **“executed”** estão relacionadas a um único evento e cada evento está relacionado ou a uma sessão *Pomodoro*, ou a um comando que foi executado. As tabelas **“tag”**, **“command_label”**, e **“status”** serem isoladas permite fácil expansão e manutenção do conteúdo dessas tabelas, dessa forma fica trivial passar a armazenar dados de um novo comando do kw por exemplo.

Otimizações e utilidades

Uma particularidade do modelo proposto é que os dados sobre cada evento estão separados em diversas tabelas, o que a princípio dificulta a inserção e consulta desses dados, já que é necessário inserir informações em várias tabelas para cada evento e fazer a junção dessas tabelas para fazer consultas sobres os dados. Apesar disso, esses problemas podem ser solucionados através do uso de algumas funcionalidades úteis para lidar com essas situações e que podem também oferecer otimizações para o funcionamento do banco de dados como um todo.

A primeira funcionalidade que pode ser utilizada para auxiliar no acesso aos dados de maneira lógica são as *visualizações* (VIEW[10]). Uma VIEW nada mais é do que um comando SELECT pré-programado, que pode ser consultado como se fosse uma tabela do banco

[10] https://www.sqlite.org/lang_createview.html

de dados. Dessa forma, pode ser criada uma *visualização* que represente o conjunto das sessões do método *Pomodoro*, gerando algo equivalente a uma tabela específica para essas sessões. Seguindo esse raciocínio, foi criada uma VIEW que agrupasse os dados referentes aos comandos do kw que foram executados para a geração das estatísticas, e outra que faz o mesmo para os dados referentes às sessões *Pomodoro*. Com isso, é possível acessar diretamente os dados necessários para geração dos relatórios e estatísticas de uso como se tivessem suas próprias tabelas. Apesar disso, as VIEWs têm suas limitações. Cada VIEW funciona como uma tabela de apenas leitura, ou seja, não é possível inserir, atualizar, ou deletar os dados de uma VIEW como pode ser feito em tabelas normais; para contornar essas limitações é possível utilizar outra funcionalidade oferecida pelo SQLite, o uso de *gatilhos* (TRIGGER[11]).

Os TRIGGERs funcionam de maneira análoga a funções que são invocadas condicionalmente, cada TRIGGER consiste de um conjunto de operações SQL que são executadas assim que alguma condição é atingida. Usando TRIGGERs e VIEWs, a interação do kw com o banco de dados fica mais linearizada, pois permite que o kw realize consultas nos conjuntos de dados que já são reconhecidos pelo sistema original de armazenamento de dados, simplificando inclusive o processo de transição do projeto.

Um outro recurso é a possibilidade de criação de índices (INDEX). Um INDEX é essencialmente uma reordenação dos dados. A maneira como os dados são armazenados no banco de dados mantém os dados ordenados em ordem crescente do número de identificação. O problema é que esse número não tem nenhuma significância implícita; trata-se apenas de um identificador aleatório, ou seja, no caso de uma busca pelo nome de uma pessoa, por exemplo, o SQLite teria que fazer uma busca linear nos dados, pois não tem como saber a qual número qualquer nome está associado. Para otimizar essa busca, é possível criar um INDEX que ordene a lista de dados a partir da ordem alfabética dos nomes por exemplo, e com isso o SQLite consegue realizar uma busca binária nos dados, que é muito mais rápida em média do que uma busca linear. Nesse sentido, foram criados INDEX que ordenam as tabelas a partir de critérios lógicos, de acordo com os dados, como ordem alfabética para as tabelas de **“tag”** e **“command_label”**, e também, criticamente, a ordem cronológica dos eventos na tabela **“event”**.

3.2.3 Contribuição em números

A implantação de um sistema de banco de dados como proposto é um passo importante para a contínua expansão e melhora da ferramenta kw. Do lado do desenvolvimento da ferramenta, o uso de um banco de dados permite o acesso a ferramentas estatísticas mais robustas já implementadas pela biblioteca, diminuindo a quantidade de código que deve ser mantido dentro do kw, já que o sistema de gerenciamento dos arquivos de dados e as funções que calculam as estatísticas de uso atualmente não são mais necessárias. Do lado dos usuários, futuramente será possível ter acesso a diferentes estatísticas sobre o uso da ferramenta que poderão ser adicionadas mais facilmente pelo kw.

O primeiro PR[12] relacionado ao banco de dados já foi incorporado ao projeto, e conta

[11] https://www.sqlite.org/lang_createtrigger.html

[12] https://github.com/kworkflow/kworkflow/pull/521

com a adição de 499 linhas distribuídas em 5 commits que implementam a interface inicial da biblioteca de acesso ao banco de dados. Esse PR foi desenvolvido ao longo de 12 dias, e gerou 36 comentários de discussão e revisão. A segunda parte da implementação do banco de dados, que inclui o modelo em si, o *script* de migração, e a integração com o resto do kw, se encontra em desenvolvimento. O PR relacionado[13] já adiciona mais de 700 linhas e remove mais de 150, e mais importante, já possui mais de 50 comentários, revisando e discutindo melhorias e integrações a serem também implementadas.

3.3 kw mail

Como apresentado na introdução, o modelo de contribuição para o kernel Linux depende do envio de *patches* por e-mail. Esse processo costuma causar algum estranhamento e confusão especialmente para pessoas que nunca contribuíram com o kernel. Isso se aplica tanto para desenvolvedores iniciantes quanto para desenvolvedores com experiência em outros projetos com modelos de contribuição diferentes.

O atual modelo de contribuição para o kernel Linux requer que as alterações propostas sejam enviadas em um arquivo que siga o formato de *patch*, que é um arquivo em *plain-text* que contém um cabeçalho com uma referência ao *commit* da mudança, o título da mudança, informações sobre o autor da mudança e a data. Após o cabeçalho, o arquivo contém uma mensagem associada ao *commit*, seguido de um resumo dos arquivos afetados pela mudança e por último um detalhamento de todas as alterações propostas. Esse arquivo então é ideal para ser enviado por e-mail para os mantenedores e listas de e-mail relevantes.

A troca de e-mails pela internet depende de alguns protocolos de internet específicos para o envio, como o *Simple Mail Transfer Protocol* (SMTP) e para o recebimento dos mesmos, como o *Internet Message Access Protocol* (IMAP). Como o foco do kw é auxiliar o desenvolvedor a enviar *patches* para o kernel, o protocolo de maior relevância para o projeto é o SMTP.

Os desenvolvedores, especialmente aqueles que não tem familiaridade com o processo de contribuição, devem se atentar para que o e-mail enviado contendo o *patch* esteja em formato *plain-text*, já que é comum que serviços de e-mail, tais como o Gmail[14], automaticamente formatem o e-mail, convertendo *tabs* por *espaços*, quebrando linhas longas, e até inserindo elementos HTML. Essa formatação é especialmente problemática por quebrar o funcionamento de diversos scripts que os mantenedores utilizam para a leitura automática desses *patches*. Nesse sentido, é muito comum, e até recomendado, que o envio desses *patches* seja feito através de uma ferramenta do Git chamada `git send-email`. Essa ferramenta tem a capacidade de formatar corretamente os *patches* e garante o envio dos e-mails com o formato correto. Porém, o uso dessa ferramenta requer que o desenvolvedor configure corretamente seu ambiente de desenvolvimento para integrar o Git com seu serviço de e-mail. Esta etapa nem sempre é intuitiva e pode causar confusão e frustração para o usuário[15], devido aos diferentes parâmetros de configuração que devem

[13] https://github.com/kworkflow/kworkflow/pull/531

[14] https://docs.kernel.org/process/email-clients.html

[15] https://lwn.net/Articles/811742/

ser selecionados.

3.3.1 Configuração do send-email

As configurações utilizadas pela ferramenta `git send-email` são armazenadas e acessadas da mesma maneira que outras configurações do sistema Git. Essas opções estão dispostas em arquivos que separam as configurações em categorias com uma série de pares chave-valor para cada uma das opções disponíveis e com isso podem ser referenciadas utilizando o padrão *<categoria>.<configuração>* por meio da ferramenta *git config*. O sistema Git ainda utiliza diferentes arquivos referentes a cada um dos escopos disponíveis, sendo eles:

- *system* que se aplica a todos os usuários da máquina
- *global* que se aplica a todos os repositórios do usuário
- *local* que se aplica apenas ao repositório atual

Devido a natureza individual da ferramenta kw, as funcionalidades que interagem com essas configurações se limitam a trabalhar nos escopos global e local, já que assim se evita que a configuração de outros usuários da mesma máquina seja alterada.

A primeira informação que a ferramenta precisa são os valores de identificação pessoal do usuário, sendo elas, nome (*user.name*) e e-mail (*user.email*). Esses valores servem apenas para identificação do usuário de maneira genérica e são usados pelo sistema Git como um todo. Além disso, é necessário a configuração das opções que serão utilizadas para realizar a conexão através do protocolo SMTP entre o `git send-email` e o provedor de e-mails do usuário. Para isso, são necessários os valores de usuário (*sendemail.smtpuser*), que é normalmente o endereço de e-mail pelo qual serão enviados os *patches* e não há necessidade de ser o mesmo que o e-mail de identificação (*user.email*), alguns serviços de e-mail usam algum identificador de usuário ao invés do endereço de e-mail de fato; o servidor de acesso SMTP (*sendemail.smtpserver*) do provedor de e-mail utilizado, bem como a respectiva porta de acesso (*sendemail.smtpserverport*). Muitas vezes, pode ser necessário também especificar o protocolo criptográfico a ser utilizado (*sendemail.smtpencryption*), assim como alguma senha de acesso do usuário (*sendemail.smtppass*).

O Programa 3.1 é um exemplo da sequência de comandos que um desenvolvedor poderia usar para realizar parte dessa configuração e o formato do arquivo de configuração resultante pode ser visto no Programa 3.2.

Programa 3.1 Configurando `git send-email`

```
1  $ git config user.name 'Rubens Gomes Neto'
2  $ git config user.email rubens.gomes.neto@usp.br
3  $ git config sendemail.smtpserver smtp.gmail.com
4  $ git config sendemail.smtpuser rubens.gomes.neto@usp.br
5  $ git config sendemail.smtpserverport 587
6  $ git config sendemail.smtpencryption tls
```

Essas configurações podem representar uma dificuldade para desenvolvedores novatos já que precisam buscar entender e encontrar os valores corretos para cada configuração,

Programa 3.2 Arquivo de configuração do Git

```
# este arquivo armazena as configurações locais
$ cat .git/config
...
[user]
   name = Rubens Gomes Neto
   email = rubens.gomes.neto
[sendemail]
   smtpserver = smtp.gmail.com
   smtpuser rubens.gomes.neto@usp.br
   smtpserverport = 587
   smtpencryption = tls
...
```

além disso mesmo desenvolvedores experientes precisam memorizar muitos desses valores, especialmente quando precisam configurar um ambiente novo. Portanto, existe aqui a possibilidade do kw auxiliar nesse processo de configuração providenciando ferramentas e utilidades que tragam benefícios tanto para desenvolvedores iniciantes, quanto para desenvolvedores experientes.

Com isso, foi implementada uma nova funcionalidade ao kw, o comando `kw mail`, como forma de superar os problemas apresentados. No âmbito prático, o `kw mail` apresenta uma CLI que pode ser utilizada para configurar cada uma das configurações de maneira individual, possuindo validação para os endereços de e-mail e com confirmação antes de alterar qualquer valor previamente configurado, ajudando a diminuir o risco de valores errados, ou mal formatados serem adicionados. As maiores vantagens dessa interface sobre a interface oferecida pelo próprio Git é a presença das validações recém citadas e também a possibilidade de configurar mais de um valor com apenas um comando, por exemplo, usando a interface do kw a configuração realizada no Programa 3.1 fica reduzida ao comando demonstrado no Programa 3.3.

Programa 3.3 Configuração `git send-email` usando o `kw mail`

```
kw mail --setup --name 'Rubens Gomes Neto' --email rubens.gomes.neto --
    smtpserver smtp.gmail.com --smtpuser rubens.gomes.neto@usp.br --
    smtpserverport 587 --smtpencryption tls

# alternativamente pode ser usada a versão reduzida da opção --setup
kw mail -t <args>
```

Um outro ponto de integração do kw com as configurações do Git se dá a partir da implementação de um *wrapper* do comando `git config` que busca nas configurações do Git do usuário os valores relevantes para o uso da funcionalidade de e-mail e os armazena em um *array* associativo que pode ser acessado por todas as utilidades dentro `kw mail`. A partir dessas informações, é possível criar a funcionalidade de listar as configurações mencionadas, mostrando os valores atualmente configurados para cada escopo acessível. Dessa forma, o usuário pode facilmente conferir seu ambiente para saber quais das configurações estão faltando ou se alguma delas não está como esperado. Além disso,

a checagem da configuração pode ser feita de maneira automatizada pelo próprio kw, de maneira a informar ao usuário quais das opções ainda precisam ser configuradas em seu ambiente ou se todos os valores já foram preenchidos. Essas funcionalidades podem também auxiliar o usuário a diagnosticar qualquer problema de configuração que possa porventura encontrar.

Apesar dessas facilidades, uma simples implementação de interface como a apresentada não simplifica de maneira substancial o processo de configuração do ambiente e pouco contribui para auxiliar os desenvolvedores novatos que ainda precisariam encontrar os valores relevantes a seu provedor de e-mail. Porém, garantem que o usuário tenha controle direto e individual de cada configuração pertinente, podendo realizar alterações desejadas sem precisar utilizar outras ferramentas. Devido a isso, com a intenção de simplificar a experiência do usuário novato é possível fazer toda a configuração de maneira interativa, onde o kw pede os valores de configuração uma a uma, guiando o usuário através de todo o processo de configuração. Uma reprodução do processo interativo pode ser visto no Programa 3.4.

Programa 3.4 Interface de configuração interativa

```
We will start with the essential configuration options!

[global] Setup your name:
  Enter new name: Rubens Gomes Neto

============================================================

[global] Setup your email:
  Enter new email: rubens.gomes.neto@usp.br

============================================================

[global] Setup your smtpuser:
  kw will set this option to rubens.gomes.neto@usp.br
  Do you want to change it? [y/N]: n

============================================================

[global] Setup your smtpserver:
  Enter new smtpserver: smtp.gmail.com

============================================================

[global] Setup your smtpserverport:
  Enter new smtpserverport: 587

============================================================

These are the optional configuration options.

[global] Setup your smtpencryption:
  Enter new smtpencryption: tls
```

cont ⟶

```
⟶ cont
33
34  ============================================================
35
36  [global] Setup your smtppass:
37   Enter new smtppass:
38
39  ============================================================
40
41    [global] 'user.name' was set to: Rubens Gomes Neto
42    [global] 'user.email' was set to: rubens.gomes.neto@usp.br
43    [global] 'sendemail.smtpuser' was set to:
44     rubens.gomes.neto@usp.br
45    [global] 'sendemail.smtpserver' was set to: smtp.gmail.com
46    [global] 'sendemail.smtpserverport' was set to: 587
47    [global] 'sendemail.smtpencryption' was set to: tls
```

A implementação desse modo interativo aguarda a coleta de todos os valores antes de consolidar as alterações, agindo de maneira atômica. Dessa forma, o processo pode ser interrompido durante a execução e o sistema do usuário não sofrerá alterações parciais. Isso é útil para o caso de falhas ou pode até mesmo ser utilizado como forma de cancelar o processo. O uso desse modo interativo simplifica as primeiras configurações do desenvolvedor já que apresenta todas as opções necessárias para o usuário, porém ainda espera que o usuário saiba quais são os valores corretos para configurações como o endereço do servidor do e-mail ou sua porta de acesso. Muitos desses valores são fixos para cada provedor de e-mail. Com isso, é possível oferecer de antemão para o usuário alguns modelos, ou *templates*, com esses valores pré-configurados para alguns provedores de e-mail.

Como o kw tem como público-alvo desenvolvedores do Kernel Linux, para decidir quais provedores de e-mail deveriam ser priorizados no oferecimento desses *templates*, foi criado um *script* (ver Programa A.1 no Apêndice A) para realizar um levantamento dos domínios de e-mails mais utilizados pelos contribuidores do kernel desde o primeiro *commit* até a tag de versão 5.13 do repositório da *mainline*[16]. Este levantamento está descrito na Tabela 3.2.

Os resultados desse levantamento mostram que o domínio mais utilizado, com quase quatro vezes mais ocorrências que o segundo colocado, é o *gmail.com*. De certa forma o domínio do Gmail nesta lista era esperado, visto que alguns dados indicam que este é um dos serviços de e-mail mais utilizados no mundo atualmente[17]. Todos os outros domínios, dentro dos 10 mais frequentes, são domínios de empresas, como AMD e Intel, que contratam desenvolvedores para desenvolver e manter os drivers de seus dispositivos. Esses domínios, porém, apresentam um desafio no quesito de criação de templates já que o provedor de e-mail interno utilizado por essas empresas não é necessariamente de conhecimento público, além de que essas empresas podem implementar servidores particulares específicos que exigiriam configurações únicas. Então, para complementar esse levantamento – e

[16] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/

[17] https://statisticsanddata.org/data/most-popular-email-providers-in-history/

Domínio	Ocorrências
gmail.com	5884
intel.com	1601
redhat.com	567
amd.com	555
google.com	541
huawei.com	454
codeaurora.org	381
ti.com	380
linux.vnet.ibm.com	326
samsung.com	319
freescale.com	266
linux.intel.com	242
linaro.org	240
marvell.com	226
broadcom.com	220

Tabela 3.2: *Domínios mais usados pelos desenvolvedores do kernel*

levando em consideração que um contribuidor novato ao desenvolvimento do kernel provavelmente não terá o mesmo perfil dos desenvolvedores veteranos – foi consultada uma lista dos provedores de e-mail mais utilizados no mundo. A partir dessas fontes e um pedido específico do mantenedor do kw de incluir um template para o provedor Riseup, foram definidos 5 templates iniciais como:

- Codeaurora
- Gmail
- Outlook
- Riseup
- Yahoo

A inclusão dos templates traz benefícios na usabilidade tanto para desenvolvedores novatos quanto para veteranos. Usuários sem experiência não precisam mais buscar quais são os valores relevantes para seu provedor de e-mail, além disso os *templates* estão integrados com a configuração interativa. Dessa forma, o usuário pode escolher algum dos *templates* disponíveis no início do processo de configuração, usando a interface mostrada no Programa 3.5, podendo pular as opções já preenchidas. No caso dos usuários mais experientes, a escolha do *template* pode ser feita por linha de comando, dessa forma é possível realizar uma configuração completa e válida usando apenas um comando. Esse comando fica muito mais reduzido e menos confuso do que o apresentado no Programa 3.3, esse comando melhorado está disposto no Programa 3.6.

Esses templates estão desacoplados do código-fonte do kw na forma de arquivos que contém os pares chave-valor relevantes. Dessa maneira, usuários futuros podem facilmente contribuir com novos templates para provedores ainda não suportados, mesmo

Programa 3.5 Interface interativa de seleção de *template*

```
1   You may choose one of the following templates to start your configuration.
2   (enter the corresponding number to choose)
3   1) Codeaurora 3) Outlook 5) Yahoo 7) Exit kw mail
4   2) Gmail 4) Riseup 6) Skip template
5   #? 2
```

Programa 3.6 Comando reduzido de configuração

```
1   kw mail --template=gmail --name 'Rubens Gomes Neto' --email 'rubens.gomes.
        neto@usp.br'
```

sem entender o funcionamento do kw. O Programa 3.7 é um exemplo de um desses arquivos.

Programa 3.7 Arquivo de *template* para o Gmail

```
1   $ cat etc/mail_templates/gmail
2   sendemail.smtpserver=smtp.gmail.com
3   sendemail.smtpencryption=tls
4   sendemail.smtpserverport=587
```

3.3.2 Envio dos e-mails

O processo de envio de e-mail com a ferramenta `git send-email` depende apenas da execução da ferramenta com os argumentos relevantes e alguma referência sobre quais *commits* do atual repositório devem ser enviados. Como a ferramenta tem uma aplicação genérica, ela possui muitas opções que alteram de diversas maneiras o seu funcionamento. Dada essa abrangência, projetos como o kernel Linux tem uma lista de argumentos que devem ser usados por padrão para o envio dos *patches*. Esses argumentos podem então gerar alguma confusão a um contribuidor novato e ainda representam mais uma lista de coisas que precisam ser memorizadas ou anotadas por cada contribuidor.

Dado o público-alvo específico do kw, as utilidades da ferramenta podem ter opções e implementações especificamente voltadas para o desenvolvimento do kernel Linux, assim é possível simplificar o comando necessário para o envio dos *patches* ao assumir algumas opções por padrão. Em particular, podemos destacar que ao enviar um conjunto de múltiplos *patches*, espera-se que o primeiro e-mail seja uma *cover-letter* que introduz e resume o *patchset* e cada *patch* subsequente seja em resposta a esse primeiro e-mail[18]. Dessa forma, a primeira iteração da interface de envio do kw mail se apoia nessas expectativas para que o comando digitado seja mais simples.

Para o envio de um *patchset* para o kernel Linux, um comando comumente usado, que segue as expectativas do projeto, costuma ter a seguinte forma:

[18] https://docs.kernel.org/process/submitting-patches.html

```
git send-email --annotate --cover-letter --thread --no-chain-reply-to --to='
    teste@email.com' --cc='mailing@list.com' -3
```

A maioria dos parâmetros utilizados são parâmetros requisitados pela documentação de contribuição do kernel[19]. Assim, uma simplificação que pôde ser implementada no `kw mail` foi o uso dessas opções como padrão para a ferramenta. Dessa forma, o comando do `kw mail` que produziria um resultado equivalente ao comando anteriormente apresentado seria:

```
kw mail --send --to='test@email.com' --cc='mailing@list.com' -3
```

Com esses exemplos, é possível ver que a interface do kw oferece uma simplificação sobre a interface padrão e para manter a flexibilidade da ferramenta, os argumentos usados como padrão podem ser alterados pelo usuário que desejar utilizar alguma outra combinação de argumentos. Adicionado a isso, por padrão a ferramenta `git send-email` aceita múltiplos valores para os campos de destinatários, desde que separados por vírgulas, e cada um desses valores pode conter o nome do destinatário junto ao seu endereço de e-mail, seguindo o seguinte formato:

```
--to='test@mail.com,Xpto Lala <xpto@mail.com>,...'
```

Esses valores não passam por validação, isso pode resultar em problemas, se o usuário utilizar o padrão de nome comum em citações, *'Sobrenome, Nome'*, quando esse padrão é utilizado, a ferramenta entende o *'Sobrenome'* como um destinatário a parte, resultando em um erro no envio que gera frustração em novatos e veteranos. Por essa razão, uma vantagem oferecida pela implementação da ferramenta `kw mail` é a validação dos endereços dos destinatários passados para os argumentos `--to` e `--cc`, buscando assim garantir a integridade e validade de todos os valores passados.

Integrando o fluxo de desenvolvimento do kernel com o kw mail

Outro aspecto importante sobre os *patches* que são enviados por e-mail para o kernel Linux são os destinatários para quem as alterações propostas devem ser encaminhadas, de acordo com a documentação do kernel os *patches* devem ser enviados para os mantenedores responsáveis pelo subsistema ao qual a proposta de alteração se refere, além disso devem ser encaminhadas cópias para as listas de e-mail relevantes. Os e-mails de todos os mantenedores, assim como as listas de e-mails dos subsistemas, se encontram em um arquivo chamado *MAINTAINERS*[20], presente na raiz da árvore de arquivos do kernel Linux. Para selecionar apenas os mantenedores e listas de e-mail relevantes para um determinado arquivo, pasta, ou arquivo de *patch*, o projeto mantém um *script* em *perl* chamado *get_maintainer.pl* que busca nesse arquivo de mantenedores pelos valores relevantes e os formata de acordo com os argumentos passados. Um exemplo de uso desse *script* pode ser visto no Programa 3.8.

Um fluxo comum de trabalho de um desenvolvedor do kernel, representado na Figura 3.4, consiste em implementar as alterações pretendidas, utilizar o *script*

[19] https://docs.kernel.org/process/5.Posting.html

[20] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/MAINTAINERS

Programa 3.8 Exemplo de usos do *get_maintainer.pl*

```
1   # comando para obter os mantenedores
2   $ perl scripts/get_maintainer.pl --nogit --nogit-fallback --nokeywords --
        norolestats --no-l drivers/gpu/drm/amd/amdgpu
3   Alex Deucher <alexander.deucher@amd.com>
4   "Christian König" <christian.koenig@amd.com>
5   "Pan, Xinhui" <Xinhui.Pan@amd.com>
6   David Airlie <airlied@linux.ie>
7   Daniel Vetter <daniel@ffwll.ch>

8   # comando para obter as listas de e-mail
9   $ perl scripts/get_maintainer.pl --nogit --nogit-fallback --nokeywords --
        norolestats --no-m drivers/gpu/drm/amd/amdgpu
10  amd-gfx@lists.freedesktop.org
11  dri-devel@lists.freedesktop.org
12  linux-kernel@vger.kernel.org
```

get_maintainer.pl para gerar a lista de destinatários por *patch*, para poder então enviar as alterações para os endereços encontrados. Esse processo pode ser tedioso e suscetível a erros, já que deve ser feito individualmente para cada *patch*. Dessa forma, esse é um dos pontos do processo de contribuição que pôde ser melhorado através da implementação de uma automação desses passos na ferramenta `kw mail`.

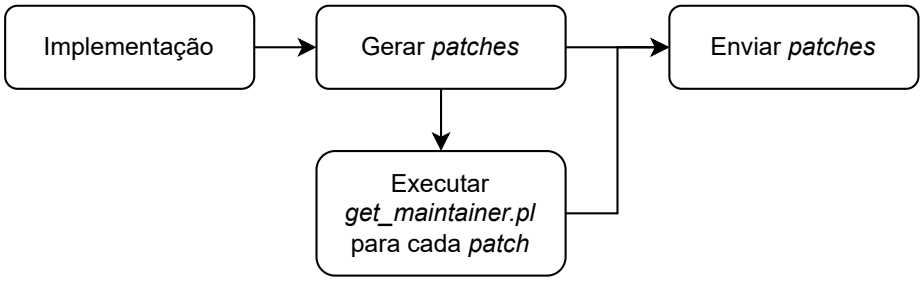


Figura 3.4: *Fluxo de envio de patches*

Uma leitura cuidadosa da documentação das opções oferecidas pela ferramenta `git send-email` revela duas opções promissoras para a implementação dessa automação, são elas `--to-cmd` e `--cc-cmd`. Essas opções recebem como parâmetro um comando que será executado uma vez para cada arquivo de *patch*, o comando receberá o caminho para o arquivo de *patch* como argumento e saída esperada desse comando é de um endereço de e-mail por linha, esses endereços são então adicionados aos destinatários diretos (`--to`), ou destinatários de cópias (`--cc`). Dessa forma, poderia ser passado através desses argumentos um comando que execute o *script get_maintainer.pl* com os argumentos corretos que gerem uma saída que esteja corretamente formatada para o `git send-email`.

Essa primeira proposta de solução funciona muito bem quando o desenvolvedor está enviando uma alteração que contém apenas um arquivo de *patch*, porém passa a apresentar problemas assim que mais de um arquivo de *patch* precisa ser enviado de uma vez, isso é conhecido como *patchset*. Para o envio de *patchsets* a documentação do kernel requer que, além das alterações presentes nos diferentes *patches*, seja também enviada uma *cover-letter*

contendo uma breve descrição e resumo das mudanças propostas pelo *patchset*. Como o arquivo da *cover-letter* não é reconhecido pelo *script get_maintainer.pl* como sendo um arquivo de *patch*, o mesmo não consegue produzir uma lista de destinatários como para os outros arquivos.

Além disso, no caso de um *patchset* que possua alterações em diferentes subsistemas é esperado que os mantenedores de todos os subsistemas afetados recebam a *cover-letter*, mas apenas os *patches* que são relevantes para o respectivo subsistema. De certa forma, a lista de destinatários, diretos e com cópia, da *cover-letter* deve ser a união das listas de destinatários de todos os *patches* do *patchset*. Ademais, a implementação do `git send-email` processa a *cover-letter* antes do que os outros *patches*, portanto não é possível utilizar um comando que acumule os endereços gerados para cada *patch* e utilize esses valores para a *cover-letter*.

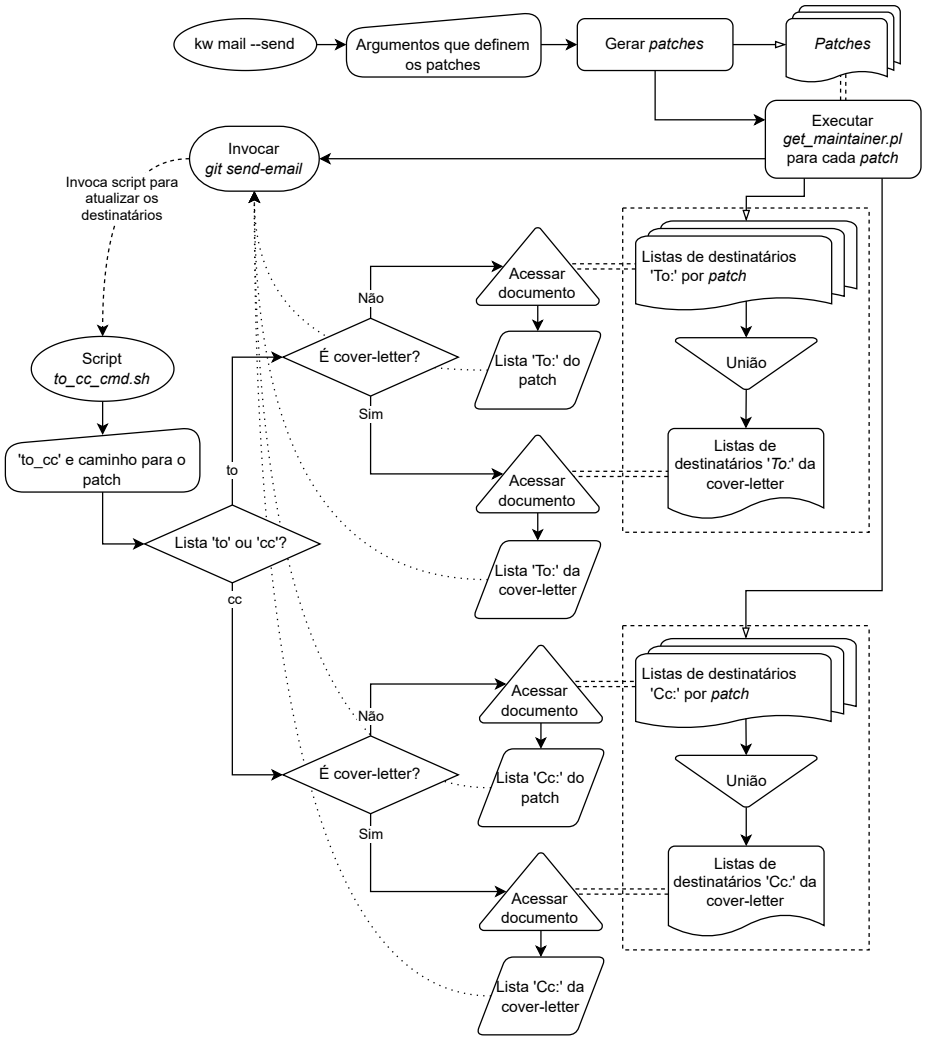


Figura 3.5: *Fluxo de execução do `kw mail`*

Para atacar diretamente esses problemas, a implementação de `kw mail` segue o fluxo de execução delineado pela Figura 3.5, ou seja, quando o comando `kw mail --send` é executado, o programa gera os arquivos de *patch* antes de invocar o `git send-email`. Dessa maneira, é possível executar o *script get_maintainer.pl* para cada um dos arquivos de

patch gerados, os endereços de e-mail encontrados são armazenados em arquivos referentes a cada *patch* na pasta de cache do kw. Ao longo desse processo, esses valores são também adicionados a um arquivo referente a *cover-letter* que ao final do processo contém a união de todos os endereços de e-mail que receberão ao menos um dos *patches*. Com isso, as opções `--to-cmd` e `--cc-cmd` do `git send-email` são usadas para invocar um *plugin* do kw que lê os valores armazenados previamente para cada *patch*, incluindo a *cover-letter*. Isso proporciona o comportamento esperado pela documentação do kernel de maneira automática e que não depende do usuário.

Programa 3.9 Envio de três *patches* para o kernel

```
$ kw mail -s -3
```

Com a atual implementação da ferramenta `kw mail`, o envio de *patches* para o kernel Linux fica simplificado. Supondo um caso em que um desenvolvedor tenha feito algumas alterações no kernel abrangendo três *commits*, de forma que o *patchset* irá conter um *patch* por *commit*, esse desenvolvedor poderá enviar suas alterações apenas utilizando o Programa 3.9.

3.3.3 Contribuição em números

Durante o desenvolvimento da funcionalidade de e-mail do kw, buscou-se melhorar a experiência do usuário desenvolvedor do kernel Linux, visto que a etapa de envio das mudanças ao kernel costuma oferece alguns desafios. Além disso, foi possível implementar funcionalidades que podem diminuir a curva de aprendizado para desenvolvedores que estão tentando começar a contribuir para o kernel. Nesse sentido, as primeiras utilidades do comando `kw mail` tratam do processo de configuração do ambiente do usuário. Para desenvolvedores experientes o `kw mail` tenta oferecer uma interface de configuração mais amigável e eficiente, já que possibilita uma rápida configuração dos valores necessários. Para desenvolvedores novatos o `kw mail` oferece uma configuração guiada que evita que o usuário tenha que buscar como realizar esta configuração manualmente. Além disso, existe a possibilidade de listar e verificar o estado da sua configuração atual.

No que diz respeito ao envio de fato dos *patches* para o kernel através do uso de e-mail, a ferramenta `kw mail` reduz drasticamente o número de passos e processos envolvidos. A atual implementação permite que qualquer desenvolvedor, seja uma pessoa experiente em contribuições para o kernel, ou alguém que está apenas começando, possa se preocupar apenas com o desenvolvimento em si, ao invés de perder tempo buscando quais os destinatários de cada *patch* e se o e-mail está formatado corretamente.

O desenvolvimento do componente `kw mail` ficou concentrada em três PRs, os dois primeiros[21] [22], focados na configuração do ambiente, continham um total agregado de 15 *commits* que adicionaram 2203 e removeram 207 linhas. Esse primeiro pacote passou por um período de mais de 70 dias de desenvolvimento, em que no total os PRs receberam mais

[21] https://github.com/kworkflow/kworkflow/pull/437

[22] https://github.com/kworkflow/kworkflow/pull/509

de 150 comentários sugerindo alterações. O último PR[23] foi dedicado a implementação do processo de envio dos e-mails. Esse PR adicionou 822 e removeu 18 linhas do projeto, angariando um total de 82 comentários ao longo de 40 dias de desenvolvimento. Parte significativa das linhas alteradas por esses PRs foram dedicadas aos testes e a documentação dessa nova funcionalidade.

[23] https://github.com/kworkflow/kworkflow/pull/535

Capítulo 4

Considerações Finais

Neste trabalho, foi apresentado parte do modelo de contribuição para o kernel Linux, que é o maior projeto colaborativo de software do mundo[1], e por isso utiliza diversos processos e ferramentas que permitem o contínuo desenvolvimento do projeto. Por esse motivo, contribuidores do kernel precisam aprender a seguir todos os processos da maneira correta e como utilizar as diferentes ferramentas oferecidas para cada uma das etapas do processo de contribuição. De forma que não existia uma ferramenta que proporcione uma experiência unificada de desenvolvimento, ou seja, que atue em todas as etapas do fluxo de desenvolvimento.

Nesse contexto, surgiu o KernelWorkflow, que tem a proposta de unificar as diferentes etapas do processo de desenvolvimento do kernel sobre uma mesma ferramenta e assim oferecer simplificações para essas diferentes etapas. Então, o objetivo deste trabalho foi de realizar contribuições ao kw para melhorar e expandir suas funcionalidades, tornando assim o processo de contribuição para o kernel mais simples e acessível.

A primeira contribuição importante foi a refatoração do esquema de documentação do projeto, que passou a ter páginas dedicadas à documentação de cada funcionalidade do kw além da padronização ao acesso dos textos de ajuda. Essa contribuição em si não afeta diretamente o processo de contribuição para o kernel Linux, mas é importante para o amadurecimento do kw, proporcionando um caráter mais profissional à ferramenta. Além disso, uma documentação completa e acessível facilita o entendimento e adoção da ferramenta por novos usuários.

Outra contribuição de destaque envolveu a implementação de um modelo de banco de dados para atualizar o método de armazenamento dos dados coletados pela ferramenta. Previamente, o kw dependia de um sistema de acesso e criação de arquivos que era implementado pela própria ferramenta, e também dependia desse sistema para realizar os cálculos estatísticos sobre os dados. Com o modelo implementado, o próprio motor do banco de dados oferece essas funcionalidades, fazendo com que a ferramenta não precise manter implementações próprias para essas funcionalidades. Uma consideração adicional sobre o sistema de banco de dados escolhido (SQLite), é que ele está disponibilizado como

[1] Kroah-Hartman, 2016.

domínio público, com seu código aberto, mas não aceita contribuições da comunidade. Portanto, não se enquadra como Software Livre, de fato. Dado que o kw é um projeto de Software Livre, pode ser vantajoso no futuro avaliar uma possível alternativa aos SQLite que esteja sobre uma licença de Software Livre.

Por fim, foi implementada a mais interessante nova funcionalidade para o kw, que pode ser utilizada através do comando `kw mail`. Essa funcionalidade possibilita o envio de *patches* por e-mail a partir do kw. Esse é um aspecto do processo de contribuição para o kernel Linux em que o kw ainda não oferecia suporte, e que causa estranhamento para contribuidores novatos e apresenta algumas dificuldades até para contribuidores mais experientes. Dessa forma, a essa funcionalidade visa simplificar o processo de envio dos *patches*, primeiro oferecendo uma configuração interativa e com *templates* para possibilitar o uso do `git send-email`. E, posteriormente, oferecendo uma interface simplificada para o envio dos *patches* por meio de opções padrão e, especialmente, o preenchimento automático dos destinatários, suavizando assim algumas das dificuldades encontradas pelos contribuidores do kernel. Inclusive, a ferramenta já foi utilizada para enviar um *patch* ao kernel, e esse *patch* já foi aceito e incorporado ao repositório do sub-sistema. Uma cópia desse *patch* foi adicionado a este trabalho como Programa B.1 (no Apêndice B).

Ao longo deste trabalho, diversas contribuições foram feitas à ferramenta kw. Este texto foi dedicado a detalhar as três contribuições consideradas de maior significância para o projeto. Ao todo, foram mais de 100 *commits* na *branch 'unstable'* do projeto, contabilizando mais de 10 mil linhas alteradas. Esse conjunto de contribuições faz do autor deste trabalho o segundo maior contribuidor do kw, atrás apenas do mantenedor original do projeto. Uma lista compreensiva de todos os PRs já integrados ao projeto, a partir deste trabalho, pode ser encontrada no Apêndice C.

Para além deste trabalho, o kw segue em estado de desenvolvimento ativo. O projeto possui uma lista de itens à serem implementados e problemas a serem corrigidos[2], e tem planos de lançar sua versão 1.0 durante o ano de 2022. Para poder alcançar esse objetivo, pretende-se continuar contribuindo e melhorando a ferramenta. Algumas contribuições já previstas envolvem a integração do comando `kw mail` com algum gerenciador de endereços. Isso possibilitará a criação de grupos de destinatários para permitir o envio de *patches* à grupos específicos, como, por exemplo, o time do desenvolvedor. Além disso, a partir do uso do banco de dados implementado, será possível aumentar os dados gerados sobre o kw, possibilitando análises estatísticas mais aprofundadas sobre seu uso. Se permitido por cada usuário, o conjunto dos dados de vários usuários pode ser usado em pesquisas científicas sobre o fluxo de trabalho de contribuidores do kernel Linux, por exemplo. Por fim, devido a toda experiência acumulada desenvolvendo e revisando contribuições para o kw, o autor deste trabalho continuará como mantenedor do projeto, ajudando na evolução da ferramenta.

[2] https://github.com/kworkflow/kworkflow/issues

Apêndice A

Scripts

Programa A.1 Script utilizado para obtenção da lista de domínios usados no kernel

```
#!/bin/bash

declare -A domains
declare domain
declare authors
declare author
declare output=''
declare path=${1:-'.'}
declare save=${2:-''}
declare -i count

authors=$(git shortlog --summary --numbered --email --no-merges | sed -E -e '
    s/^.*@//g' -e 's/>$//g')

while read -r author; do
  if [[ "$author" =~ [[:space:]] ]]; then
    continue
  fi
  count=${domains["$author"]:-1000000000}
  domains["$author"]=$((count + 1))
done <<< "$authors"

for domain in ${!domains[@]}; do
  output+="${domains[$domain]},$domain"$'\n'
done

if [[ -n "$save" ]]; then
  printf '%s' "$output" | sort -d -r | head -100 | sed -E -e 's/^10+//g' > "
      $path"/authors_list.txt
  return 0
fi

printf '%s' "$output" | sort -d -r | head -100 | sed -E -e 's/^10+//g'
```

Apêndice B

Patches

Programa B.1 Primeiro *patch* enviado para o kernel Linux usando a ferramenta `kw mail`

```
1   From: Magali Lemes <magalilemes00@gmail.com>
2   To: harry.wentland@amd.com, sunpeng.li@amd.com,
3    Rodrigo.Siqueira@amd.com, alexander.deucher@amd.com,
4    christian.koenig@amd.com, Xinhui.Pan@amd.com, airlied@linux.ie,
5    daniel@ffwll.ch
6   Cc: Magali Lemes <magalilemes00@gmail.com>,
7    kernel test robot <lkp@intel.com>,
8    amd-gfx@lists.freedesktop.org, dri-devel@lists.freedesktop.org,
9    linux-kernel@vger.kernel.org
10  Subject: [PATCH] drm/amd/display: Use NULL pointer instead of plain integer
11  Date: Wed, 2 Feb 2022 18:38:56 -0300 [thread overview]
12  Message-ID: <20220202213856.409403-1-magalilemes00@gmail.com> (raw)
13
14  Assigning 0L to a pointer variable caused the following warning:
15
16  drivers/gpu/drm/amd/amdgpu/../display/dc/dml/dsc/rc_calc_fpu.c:71:40:
17  warning: Using plain integer as NULL pointer
18
19  In order to remove this warning, this commit assigns a NULL pointer to
20  the pointer variable that caused this issue.
21
22  Reported-by: kernel test robot <lkp@intel.com>
23  Signed-off-by: Magali Lemes <magalilemes00@gmail.com>
24  ---
25   drivers/gpu/drm/amd/display/dc/dml/dsc/rc_calc_fpu.c | 2 +-
26   1 file changed, 1 insertion(+), 1 deletion(-)
27
28  diff --git a/drivers/gpu/drm/amd/display/dc/dml/dsc/rc_calc_fpu.c b/drivers/
        gpu/drm/amd/display/dc/dml/dsc/rc_calc_fpu.c
29  index ec636d06e18c..ef75eb7d5adc 100644
30  --- a/drivers/gpu/drm/amd/display/dc/dml/dsc/rc_calc_fpu.c
31  +++ b/drivers/gpu/drm/amd/display/dc/dml/dsc/rc_calc_fpu.c
32  @@ -68,7 +68,7 @@ static void get_qp_set(qp_set qps, enum colour_mode cm,
        enum bits_per_comp bpc,
```

cont ⟶

```
    ⟶ cont
33      int sel = table_hash(mode, bpc, max_min);
34      int table_size = 0;
35      int index;
36    - const struct qp_entry *table = 0L;
37    + const struct qp_entry *table = NULL;
38
39      // alias enum
40      enum { min = DAL_MM_MIN, max = DAL_MM_MAX };
41    --
42    2.25.1
```

O *patch* pode ser acessado em seu contexto na lista de e-mail através do link: https://lore.kernel.org/lkml/20220202213856.409403-1-magalilemes00@gmail.com/T/

Apêndice C

PRs integrados ao kw

A lista a seguir representa o conjunto de todos os PRs que foram aceitos e incorporados ao kw até o dia 24 de Fevereiro de 2022. A lista segue em ordem cronológica de criação do PR.

- src: etc: update network setup for QEMU
  - https://github.com/kworkflow/kworkflow/pull/274
- setup: Make setup show usage when incorrect number of args given
  - https://github.com/kworkflow/kworkflow/pull/277
- Move ssh functionality to remote.sh
  - https://github.com/kworkflow/kworkflow/pull/281
- src: kwlib: improve path manipulation
  - https://github.com/kworkflow/kworkflow/pull/310
- doc: refactor and improve how to contribute
  - https://github.com/kworkflow/kworkflow/pull/316
- src: mk: run make olddefconfig before building
  - https://github.com/kworkflow/kworkflow/pull/339
- tests: rename utils
  - https://github.com/kworkflow/kworkflow/pull/340
- shellcheck: address SC2164
  - https://github.com/kworkflow/kworkflow/pull/341
- Man per feature
  - https://github.com/kworkflow/kworkflow/pull/348
- Improve ‘zsh‘ compatibility and setup

  - https://github.com/kworkflow/kworkflow/pull/365
- src: diff: fix diff functionality
  - https://github.com/kworkflow/kworkflow/pull/387
- Refactor init
  - https://github.com/kworkflow/kworkflow/pull/405
- explore: fix explore help text
  - https://github.com/kworkflow/kworkflow/pull/421
- doc: documentation standardization and small fixes
  - https://github.com/kworkflow/kworkflow/pull/423
- src: kw sendemail support WIP
  - https://github.com/kworkflow/kworkflow/pull/425
- kw_config_loader refactor
  - https://github.com/kworkflow/kworkflow/pull/429
- doc: include IRC server in our documentation
  - https://github.com/kworkflow/kworkflow/pull/435
- run shellcheck on push
  - https://github.com/kworkflow/kworkflow/pull/436
- Sendemail setup
  - https://github.com/kworkflow/kworkflow/pull/437
- time_and_date: remove cal dependency
  - https://github.com/kworkflow/kworkflow/pull/438
- echo -> printf
  - https://github.com/kworkflow/kworkflow/pull/439
- remove left over echo
  - https://github.com/kworkflow/kworkflow/pull/443
- build_test: add #!/bin/bash
  - https://github.com/kworkflow/kworkflow/pull/448
- report: small refactor
  - https://github.com/kworkflow/kworkflow/pull/449
- run_tests: fix run_tests bugs
  - https://github.com/kworkflow/kworkflow/pull/451

- setup: break dependencies into lines
  - https://github.com/kworkflow/kworkflow/pull/457
- doc: index: remove duplicate entry
  - https://github.com/kworkflow/kworkflow/pull/459
- setup: doc: integrate dependencies with docs
  - https://github.com/kworkflow/kworkflow/pull/463
- src: deploy: fix shellcheck 2076
  - https://github.com/kworkflow/kworkflow/pull/477
- src: kw_time_and_date: fix octal bug
  - https://github.com/kworkflow/kworkflow/pull/478
- Sendemail fixes
  - https://github.com/kworkflow/kworkflow/pull/485
- src: config_manager: rework options parser
  - https://github.com/kworkflow/kworkflow/pull/489
- src: debug: removes execute permission from file
  - https://github.com/kworkflow/kworkflow/pull/490
- src: tests: remove reintroduced echos
  - https://github.com/kworkflow/kworkflow/pull/494
- docs: dependencies: add missing send-email deps
  - https://github.com/kworkflow/kworkflow/pull/496
- kw: fix help text and man page in main
  - https://github.com/kworkflow/kworkflow/pull/503
- Mail setup using templates
  - https://github.com/kworkflow/kworkflow/pull/509
- src: db: introduce db interface
  - https://github.com/kworkflow/kworkflow/pull/521
- Move kernel scripts parameters to kworkflow.config
  - https://github.com/kworkflow/kworkflow/pull/529
- src: small optimization when using cat
  - https://github.com/kworkflow/kworkflow/pull/533
- Mail send

    - https://github.com/kworkflow/kworkflow/pull/535
- src: kw_time_and_date: fix get_week_beginning_day
    - https://github.com/kworkflow/kworkflow/pull/539
- src: kwlib: improve command_exists check
    - https://github.com/kworkflow/kworkflow/pull/541
- Sendemail setup fixes
    - https://github.com/kworkflow/kworkflow/pull/561

Referências

[Cirillo 2009] F. Cirillo. *The Pomodoro Technique*. Creative Commons, 2009. isbn: 9781445219943. url: https://books.google.com.br/books?id=ThkbQwAACAAJ (citado na pg. 13).

[*CodingGuidelines* 2021] *CodingGuidelines*. Git. url: https://github.com/git/git/blob/master/Documentation/CodingGuidelines (acesso em 01/12/2021) (citado na pg. 11).

[Haerder e Reuter 1983] Theo Haerder e Andreas Reuter. "Principles of transaction-oriented database recovery". Em: *ACM Comput. Surv.* 15.4 (dez. de 1983), pgs. 287–317. issn: 0360-0300. doi: 10.1145/289.291. url: https://doi.org/10.1145/289.291 (citado na pg. 15).

[Kroah-Hartman 2016] Greg Kroah-Hartman. *Linux Kernel Development, Greg Kroah-Hartman - Git Merge 2016*. 2016. url: https://www.youtube.com/watch?v=vyenmLqJQjs (acesso em 12/12/2021) (citado na pg. 33).

[Kroah-Hartman 2018] Greg Kroah-Hartman. *Linux Kernel Development*. 2018. url: https://github.com/gregkh/kernel-development/blob/bd8d3673b33fa641d06046fa4bff103f78ec4e89/kernel-development.pdf (acesso em 12/12/2021) (citado na pg. 2).

[Kroah-Hartman 2022] Greg Kroah-Hartman. *Linux Kernel Statistics*. 2022. url: https://github.com/gregkh/kernel-history (acesso em 10/01/2022) (citado na pg. 1).

[Linux kernel community 2022] Linux kernel community. *How the development proccess works*. The kernel development community. 2022. url: https://docs.kernel.org/process/2.Process.html (acesso em 05/01/2022) (citado na pg. 1).

[Stewart *et al.* 2020] Kate Stewart, Shuah Khan e Daniel M German. *2020 Linux Kernel History Report*. 2020. url: https://www.linuxfoundation.org/wp-content/uploads/2020_kernel_history_report_082720.pdf (acesso em 14/02/2022) (citado na pg. 1).