ESTUDO DA EVOLUÇÃO DO KERNEL LINUX ATRAVÉS DA ANÁLISE DE COMMITS

Carlos Eduardo Ferreira Reis

Orientador: Prof. Dr. Tássio Ferenzini Martins Sirqueira.

Engenharia de Software - Universidade de Vassouras Av. Expedicionário Oswaldo de Almeida Ramos, nº 280, Centro - Vassouras/RJ - Brasil

carloskinoko@gmail.com

ABSTRACT

This study is an in-depth presentation of the Linux kernel commit history on GitHub. The main factors that will be used will be the most active and contributing developers and their distribution over time. Among the main aspects investigated are the most frequently modified files, such as changes in specific periods, and how certain developers specialize in specific areas of the kernel, such as drivers, memory subsystems, or critical hardware components. In addition, the study examines the interrelationship between kernel development and external factors, such as the demand for new devices and security patches, which directly influence the areas that receive the most attention in the project. Throughout the work, external factors that influence kernel development are also discussed, such as corporate contributions, pressure for technological innovation, and the need to maintain system security and stability. The methodology used involves protecting commits via Git, converting them into a structured format for analysis, allowing the use of data science tools, such as Python and associated libraries, to extract patterns and information that would be difficult to extract otherwise.

RESUMO

Este Estudo é uma apresentação aprofundada do histórico de commits do kernel do linux no github. Os principais fatores que serão utilizados, serão os desenvolvedores mais ativos e com maior contribuição e a sua distribuição ao longo do tempo. Entre os principais aspectos investigados estão os arquivos mais frequentemente modificados, as mudanças em períodos específicos, e a forma como determinados desenvolvedores se especializam em áreas distintas do kernel, como drivers, subsistemas de memória, ou componentes críticos de hardware. Além disso, o estudo examina a inter-relação entre o desenvolvimento do kernel e fatores externos, como a demanda por novos dispositivos e patches de segurança, que influenciam diretamente as áreas que recebem mais atenção no projeto. Ao longo do trabalho, são discutidos também fatores externos que influenciam o desenvolvimento do kernel, como contribuições corporativas, pressões por inovação tecnológica e a necessidade de manter a segurança e a estabilidade do sistema. A metodologia utilizada envolve a extração dos commits via Git, convertendo-os em um formato estruturado para análise, permitindo o uso de ferramentas de ciência de dados, como Python e bibliotecas associadas, para extrair padrões e informações que seriam difíceis de se extrair de outra forma.

Palavras-Chave: Commits, Kernel Linux, Github, Contribuições, código

1 Introdução

O Linux, hoje, é um dos sistemas operacionais mais utilizado mundialmente, presente em servidores, dispositivos móveis e outros sistemas além do computador de mesa devido à sua robustez, flexibilidade e à natureza *open-source* e *free-software*[1]. O kernel do Linux, que é o núcleo do sistema, tem uma complexidade considerável e está em constante evolução, devido à colaboração de sua grande comunidade de desenvolvedores. Dado seu tamanho e importância, entender o fluxo de desenvolvimento, a evolução do código e os padrões de colaboração dentro do kernel é significativo. O núcleo do linux, é uma parte muito importante dos sistemas que está presente, pois é responsável pelo gerenciamento de recursos do sistema, como CPU, memória, armazenamento e periféricos, devido a sua complexidade técnica e da necessidade de suporte de muitas arquiteturas distintas, fazendo com que exista uma necessidade de se adaptar constantemente com as mudanças de softwares e hardwares do mercado, tornando o acompanhamento de sua evolução determinante, para aqueles que querem trabalhar com ou entender o seu fluxo.

A manutenção e o desenvolvimento do kernel do Linux são altamente colaborativos devido a sua natureza *open-source*[2], com contribuições vindo de diversos desenvolvedores. Esses desenvolvedores pertencem a diferentes perfis, desde contribuidores independentes até profissionais que trabalham em grandes empresas de tecnologia, como Intel, Red Hat, Samsung e IBM, entre outras. Cada contribuição ao kernel, seja para corrigir um bug, melhorar o desempenho ou adicionar suporte a um novo hardware, é registrada em um *commit* no sistema de controle de versão *Git*, utilizado para gerenciar o código-fonte do kernel.

A análise do histórico de *commits* pode fornecer um entendimento importante sobre a maneira como o kernel se desenvolveu ao longo do tempo. Analisar aspectos como os arquivos mais modificados, as contribuições mais relevantes e os desenvolvedores mais ativos permitem não apenas entender o presente estado do projeto, mas também prever desafios futuros. Este estudo, ao usar ciência de dados aplicada ao histórico de *commits*, busca trazer clareza sobre essas questões, dando uma perspectiva clara das áreas mais dinâmicas do desenvolvimento do kernel.

---

[1]Linux Solutions, Uso do linux ao redor do mundo.
disponível em: https://linuxsolutions.com.br/o-uso-do-linux-no-mundo-atual-saiba-como-funciona/
[2] Lupa Charleaux, como funciona o open source. disponível em:
https://www.tecmundo.com.br/software/215130-open-source-funciona.htm

2 Fundamentação teórica

Neste capítulo são abordados os principais assuntos que representam o conhecimento necessário na literatura, destacando a revisão sistemática que concedeu o suporte na revisão de artigos, livros, textos etc., que fundamentam o projeto.

2.1 Revisão da literatura

Para ser feita a revisão sistemática do presente trabalho foi utilizada a seguinte *string* de busca:

("Linux kernel" OR "Linux OS") AND "development" AND ("commit history" OR "version control" OR "source code changes" OR "software repository") AND ("collaborative development" OR "software evolution")

No qual conseguimos como resultado de busca 336 artigos, textos, trabalhos ou documentos, que passaram por uma avaliação de relevância com o tema proposto.

Tabela 1 - Critérios da Revisão Sistemática

Critério	Descrição
Seleção de Fontes	Fontes evidenciadas em base online pública
Palavras-chave	Linux kernel, desenvolvimento, histórico de commits, desenvolvimento de software
Idioma dos Estudos	Inglês - US
Métodos de busca de fontes	Pesquisas via web, excluindo buscas manuais
Listagem de fontes	Pismin, ResearchGate
Tipo dos Artigos	Tccs e Artigos.
Critérios de Inclusão e Exclusão de Artigos	Os artigos devem ser livres de cobrança, cadastro de contas e login para visualização e estarem disponíveis na web. artigos devem ser relevantes e relacionados ao assunto abordado.

Fonte: Elaborado pelo Autor

2.2 Desenvolvimento de Software Aberto

O modelo de desenvolvimento de software aberto é caracterizado por uma comunidade de desenvolvedores distribuída globalmente, que contribui para a evolução do projeto de forma coletiva. A análise histórica das contribuições em repositórios git, como o do kernel Linux, permite entender como decisões colaborativas influenciam a estabilidade e a inovação no software. Trabalhos clássicos, como os de Eric S. Raymond ("A Catedral e o Bazar") e a pesquisa de Mockus, Fielding e Herbsleb (2002), detalham os aspectos organizacionais e técnicos do desenvolvimento de software de código aberto, incluindo o fluxo de trabalho, os papéis dos contribuidores e as dinâmicas sociais.

Além disso, o Linux também é considerado um código livre, isto se dá ao fato de que seu código-fonte é aberto e disponível a todos, o que mais uma vez reforça sua natureza colaborativa, visto que qualquer desenvolvedor é capaz de contribuir de alguma forma.[3]

2.3 Mineração de dados

A mineração de dados de repositórios de software (Software Repository Mining) é um campo que se dedica a extrair conhecimento dos históricos de commits para entender a evolução do software. Ferramentas de mineração possibilitam a extração de métricas quantitativas, como número de inserções, deleções e modificações de arquivos. Isso facilita a identificação dos padrões de contribuição, a distribuição de mudanças entre autores e a evolução das diferentes partes do código. Estudos como os de Godfrey e Tu (2000) mostram como a mineração de repositórios permite detectar anomalias, identificar riscos de qualidade e mapear tendências de crescimento do software.

2.4 Métricas de Qualidade e Manutenibilidade de Código

A análise quantitativa do código é importante para avaliar a qualidade e manutenibilidade de um projeto de software. Métricas como número de inserções e deleções e frequência de alterações indicam áreas de instabilidade ou complexidade crescente. Esses aspectos foram abordados em trabalhos de Lehman (1980) sobre leis de evolução do software e continuam relevantes em estudos de sustentabilidade e segurança em sistemas complexos. A análise das contribuições por autor, como quantidade e relevância das modificações, também reflete o impacto de cada desenvolvedor no projeto, o que contribui para estratégias de manutenção e alocação de esforços.

2.5 Fundamentação

---

[3] Emanuel Negromonte, Software Livre e como o linux se encaixa. disponível em: https://sempreupdate.com.br/linux/introducao-ao-software-livre-o-que-e-e-como-o-linux-se-encaixa/

Este trabalho usou como referência diversos estudos e obras literárias, indo de textos relacionados à evolução do Linux, open source de modo geral, e até sobre chamadas de sistema, materiais que serviram como referência e aprendizado, no processo de criação deste texto.

Estudos prévios sobre o desenvolvimento do kernel Linux fornecem uma base teórica sólida para esta análise. Godfrey e Tu (2000) destacaram o crescimento geométrico do kernel, como o código seguia as leis de Lehman para seu crescimento. Outros estudos, como *Evolution of a specific part of the Linux kernel* de Tsai et al. (2016) e *Faults in Linux: ten years later* de Palix et al. (2011), abordaram a evolução de partes específicas do kernel, tais como características de APIs e a taxa de falhas de drivers, mas não trata especificamente sobre os commits do kernel em si. Contudo, *A Study of the Linux Kernel Evolution* de Oded Koren(2006), faz uma análise mais aprofundada, detalhando aspectos de versionamento, número de linhas de código, tamanho do kernel e o impacto do modelo de desenvolvimento em estilo "bazaar" sobre o crescimento e a diversidade de contribuições. Este estudo destaca a preferência pelas versões instáveis no que se refere ao aumento de linhas de código e ao crescimento do kernel, sugerindo que as contribuições mais experimentais e rápidas influenciaram a evolução do Linux.

No entanto, apesar da abrangência, o estudo de Koren já é considerado antigo, pois cobre versões até 2005. Desde então, o kernel passou por mudanças significativas, especialmente com a entrada de novos players corporativos e a demanda crescente por segurança e modularidade. Este texto traz uma semelhança maior ao texto de Oded Koren, por ser também um relacionado a evolução do Linux como um todo, porém as principais diferenças são o período em que essas informações foram extraídas, a fonte de onde veio cada informação e também aos pontos trazidos em cada texto, este texto dá uma ênfase aos desenvolvedores e alterações de arquivos no geral.

3. Materiais e Métodos

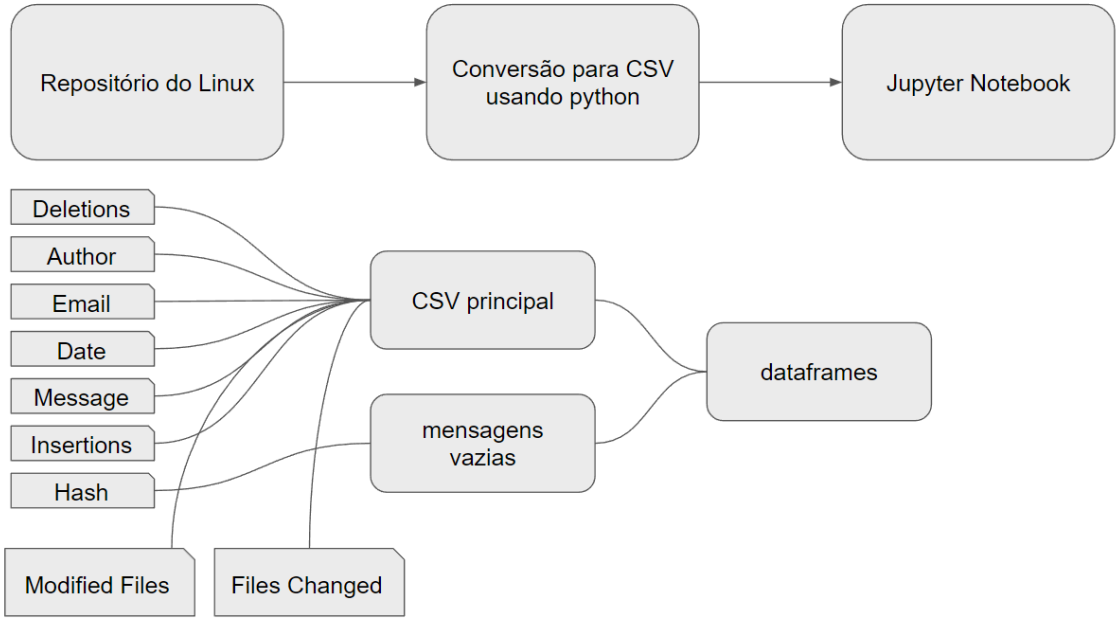
O presente trabalho foi desenvolvido em um *Jupyter Notebook*, neste capítulo serão abordadas as tecnologias escolhidas para abordar a pesquisa como um todo, desde as ferramentas e métodos para extrair os *commits*, tanto para as bibliotecas utilizadas para gerar gráficos e filtrar informações.

As tecnologias abordadas servem para facilitar o processo de extração de informações vindo de um sistema de controle de versões, cujo objetivo é servir como uma análise e referência sobre as informações que podem ser obtidas ao extrair informação através de *commits*.

3.1 Coletando informação

Para realizar a análise, foi utilizado o comando git log, pois a utilização do *pydriller* não havia gerado resultados depois de muito tempo processando, além disso, também foi utilizado do WSL do *windows* para emular o sistema do linux, pois dentro do repositório existiam palavras que não eram permitidas dentro do *windows*, o que fazia com o que o processo também não gerasse resultados. Os dados foram exportados para um formato CSV e analisados utilizando bibliotecas de ciência de dados, como Pandas e *Matplotlib*, para a criação de gráficos e identificação de padrões dentro de um *jupyter notebook*.

Figura 1: visão geral da coleta de dados.



Fonte: Elaborado pelo Autor

A figura 1 apresenta o processo geral da coleta e organização de informações. Além disso, havia um grande número de *commits* “vazios” o que poderia ser um erro durante a geração do CSV, devido ao seu tamanho, ou também relacionado aos métodos que utilizamos para o processo como um todo.

A princípio, parecia um erro, porém mesmo depois de separar os *commits* que estavam vazios em um CSV separado, não existia uma diferença muito clara entre as informações que recebidas, então foram utilizados alguns comandos de git para identificar o que existia dentro de um dos *hashes* vazios. O que acabou por ser apenas um grande número de merge *tags* e merge *branches*, sendo esses, em sua maioria feitos por Linus Torvalds (38353) até o período de 9 de setembro de 2009 seguido por David S. Miller (11575), sendo este o último *commit* feito em 9 de setembro de 2015. Essas merge tags, afetam em sua maioria, dados que utilizam o “files changed” de seus respectivos dataframes, então em sua maioria eles continuavam durante as análises.

3.2 Jupyter Notebook

O Jupyter Notebook é uma ferramenta interativa que facilita o desenvolvimento e a visualização de códigos, especialmente para análise de dados e machine learning. Inicialmente desenvolvido como parte do projeto IPython, o Jupyter foi lançado em 2014 e hoje é uma das plataformas mais usadas por cientistas de dados[4], pesquisadores e desenvolvedores. Ele permite que o usuário escreva e execute código, visualize dados, e documente o processo de análise em uma única interface. Essa combinação de código e documentação é especialmente útil para projetos que requerem análise iterativa e experimentação.

Jupyter Notebook é uma ferramenta de código aberto que utiliza notebooks interativos, onde o usuário pode combinar código Python (ou outras linguagens) com texto explicativo, gráficos e visualizações. Cada notebook é composto por células de código e células de texto, o que permite uma abordagem modular e organizada para o desenvolvimento. A estrutura baseada em células facilita a divisão do código em blocos independentes, promovendo maior clareza e reutilização.

Além do Python, Jupyter Notebook suporta diversas outras linguagens por meio de extensões chamadas kernels, como R, Julia e SQL, permitindo que desenvolvedores escolham a linguagem que melhor se adapta ao projeto. Essa flexibilidade é uma das razões que tornam o Jupyter tão popular no meio acadêmico e em empresas de tecnologia, pois permite que equipes multidisciplinares colaborem de forma eficiente.

Uma das principais vantagens do Jupyter Notebook é a visualização de dados em tempo real. Com ele, é fácil integrar bibliotecas de visualização, como Matplotlib, Seaborn e Plotly, permitindo que gráficos e outras representações visuais de dados sejam gerados e exibidos no mesmo ambiente. Isso torna o processo de análise de dados mais intuitivo, pois o usuário pode ver o impacto de cada alteração no código diretamente nos gráficos e relatórios.

Além disso, o Jupyter é amplamente usado para compartilhar projetos e resultados com outras pessoas. Os notebooks podem ser salvos em formato HTML ou PDF e compartilhados facilmente, seja em plataformas de controle de versão como o GitHub ou em serviços de nuvem, como o Google Colab. Isso faz com que o Jupyter Notebook seja uma excelente escolha para criar documentos que combinam código, análise e explicações, sendo ideal para apresentações, tutoriais e relatórios de pesquisa.

---

[4] Datacamp, ferramentas para ciência de dados. disponível em: https://www.datacamp.com/pt/blog/top-data-science-tools

O Jupyter notebook foi utilizado por questões de familiaridade e simplicidade, e também por receber python e outras ferramentas utilizadas sem nenhum problema.

3.3 Python

Python é uma das linguagens de programação mais populares e versáteis da atualidade[5], amplamente usada em desenvolvimento web, ciência de dados, machine learning e automação. Criada em 1991 por Guido van Rossum, Python é conhecida por sua sintaxe simples e intuitiva, que facilita a leitura e escrita de código, tornando-a acessível tanto para iniciantes quanto para programadores experientes. A linguagem permite um desenvolvimento ágil, com suporte a múltiplos paradigmas de programação, como programação orientada a objetos e funcional.

Neste projeto, Python foi utilizado por possuir muitas bibliotecas de fácil acesso e utilização, e também por permitir a utilização do Pandas, que foi uma biblioteca muito utilizada no desenvolvimento.

3.4 Pandas

Pandas é uma biblioteca de código aberto em Python que facilita a manipulação e análise de dados estruturados. Desenvolvida por Wes McKinney em 2008, Pandas fornece duas estruturas de dados principais: dataframes e Séries. O dataframe é uma tabela bidimensional que permite organizar os dados em linhas e colunas, similar a uma planilha, enquanto a Series representa uma única coluna. Essas estruturas tornam o trabalho com dados mais eficiente.

Pandas, neste projeto, foi muito utilizado para manipulação de dados, e análise, além de ser uma biblioteca exclusiva de python, que foi a escolha da linguagem principal do projeto.

3.5 Matplotlib

Matplotlib é uma das opções de biblioteca para visualização de dados em Python. Criada em 2003 por John D. Hunter, Matplotlib permite a criação de gráficos estáticos em 2D, como gráficos de linhas, dispersão, barras e histogramas. Ela oferece uma ampla gama de opções para personalizar cada aspecto dos gráficos, incluindo rótulos, cores, estilos de linhas e legendas. Essa flexibilidade torna o Matplotlib adequado para criar gráficos altamente personalizados, facilitando a comunicação visual de resultados e insights.

---

[5] PYPL, linguagens mais utilizadas. disponível em: https://pypl.github.io/PYPL.html

Matplotlib é importante para cientistas de dados e analistas, pois permite transformar dados em representações visuais que facilitam a análise e interpretação. Além disso, serve como base para bibliotecas de visualização mais avançadas, como Seaborn, que é voltada para gráficos estatísticos, e Plotly, que permite a criação de gráficos interativos. Mesmo com a concorrência dessas bibliotecas, Matplotlib continua sendo uma escolha popular devido à sua estabilidade e versatilidade.

Por meio da integração com Pandas, é possível gerar gráficos diretamente de dataframes, simplificando o processo de visualização de dados. Essa compatibilidade permite que Matplotlib seja usado ao longo do fluxo de trabalho de análise, desde a exploração dos dados até a criação de relatórios e apresentações.

Matplotlib foi utilizado para a criação de gráficos, que foram utilizados durante as análises.

3.6 Integração entre Python, Pandas e Matplotlib

Python, Pandas e Matplotlib são frequentemente usados em conjunto para formar um fluxo de trabalho completo de análise e visualização de dados. Cada uma dessas ferramentas possui um papel específico:

- **Python** fornece a base, integrando bibliotecas e pacotes para construir o pipeline de análise.
- **Pandas** organiza e manipula os dados em estruturas mais organizadas, facilitando operações complexas, como agregações, filtragens e fusões.
- **Matplotlib** permite transformar os dados em gráficos, ajudando a visualizar a informação .

Essas ferramentas formam um ecossistema consistente ,o que faz com que, analistas de dados e desenvolvedores que buscam um ambiente unificado para a análise e apresentação de dados possam trabalhar em um ambiente conciso. A combinação de Python com Pandas e Matplotlib permite um fluxo de trabalho que vai da coleta e preparação dos dados até a visualização final, oferecendo uma solução completa para projetos de ciência de dados, pesquisa e desenvolvimento de software analítico.

3.7 Comandos do Git

Git é um sistema de controle de versão distribuído e utilizado para gerenciar o histórico de alterações em projetos de software. Desenvolvido por Linus Torvalds em 2005, o Git facilita a colaboração entre desenvolvedores e permite que eles trabalhem em paralelo sem risco de sobrescrever o trabalho uns dos outros. Os comandos Git permitem aos usuários criar, modificar e controlar versões do código

de maneira eficiente, o que o torna uma ferramenta importante em equipes de desenvolvimento.

Os comandos do Git foram utilizados para a criação do CSV, e também para investigar algumas hashes específicas durante o desenvolvimento.

3.8 Windows Subsystem for Linux (WSL)

Windows Subsystem for Linux (WSL) é uma funcionalidade do Windows que permite executar um ambiente Linux completo diretamente no Windows, sem a necessidade de uma máquina virtual ou dual-boot. A Microsoft introduziu o WSL para facilitar o uso de ferramentas e comandos de desenvolvimento Linux em um sistema Windows, uma solução especialmente útil para desenvolvedores que trabalham em ambientes de desenvolvimento mistos.

O WSL foi utilizado pois esta pesquisa foi feita principalmente no windows, e o repositório do linux tem algumas peculiaridades que não permite que ele seja clonado no windows sem a utilização de uma ferramenta específica, tais como arquivos que possuem nomes reservados no windows, ou arquivos que passam do limite de caracteres.

4. Desenvolvimento do trabalho

Esta seção contém todo o desenvolvimento do trabalho, desde o código em si e suas finalidades, mostrando através de gráficos e tabelas o que foi feito utilizando os conceitos apresentados anteriormente.

4.1 Arquivos modificados

Tabela 2 - Arquivos com mais modificações

Files Changed	Modified Files	Insertions	Deletions
MAINTAINERS	7015	34146	18464
sound/pci/hda/patch_realtek.c	1934	37821	19082
drivers/gpu/drm/i915/intel_display.c	1934	25124	18835
fs/io_uring.c	1735	24688	20328
Makefile	1462	3844	3202

Fonte: Elaborado pelo Autor

Os arquivos com maior número de modificações no kernel Linux refletem áreas que recebem mais atenção, seja isso para a manutenção, compatibilidade ou evolução desse sistema. O arquivo MAINTAINERS, por exemplo, desempenha um papel na organização do desenvolvimento colaborativo, pois documenta quem é

responsável por cada subsistema do kernel. A necessidade de atualizar constantemente esse arquivo, com milhares de modificações ao longo do tempo, evidencia a importância de manter uma estrutura de gestão de responsabilidades bem definida, facilitando a coordenação entre equipes que trabalham em diferentes partes do código.

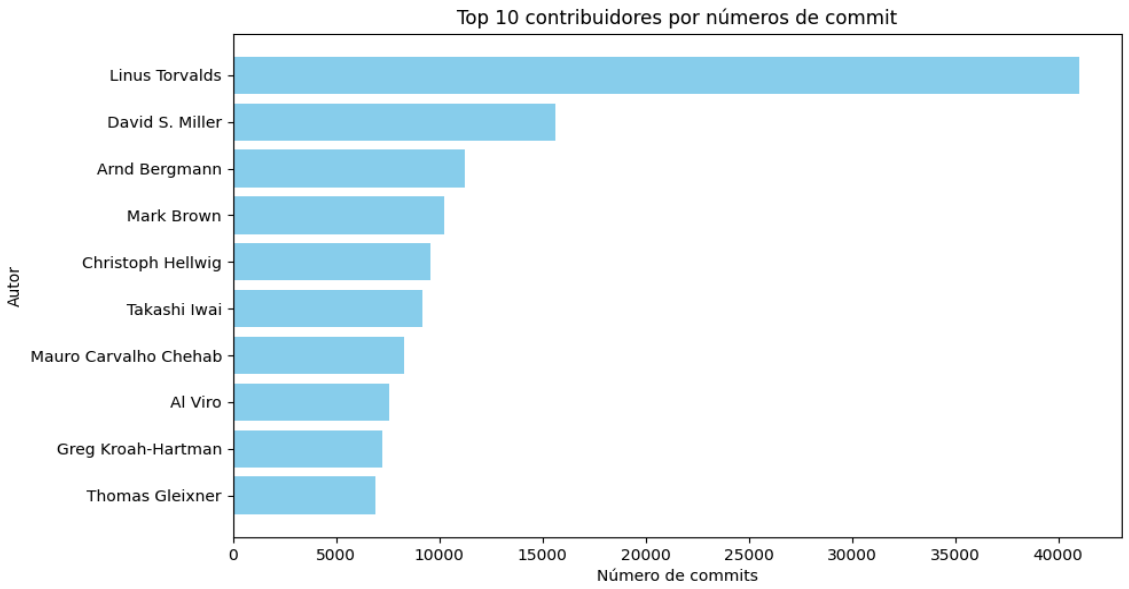
Além da organização interna, a adaptação do kernel a diferentes hardwares e tecnologias é visível nas frequentes mudanças em arquivos como patch_realtek.c, responsável por drivers de áudio da Realtek, e intel _display.c, que integra o suporte a dispositivos gráficos da Intel. Esses drivers demandam atenção constante para manter compatibilidade com novos dispositivos e otimizar o desempenho de áudio e vídeo, que são componentes que recebem bastante atenção por receberem muitas mudanças no decorrer dos anos. As constantes inserções e deleções nesses arquivos representam ajustes, correções e atualizações que acompanham o avanço do hardware e a necessidade de oferecer suporte para os dispositivos mais modernos e manter a compatibilidade com os antigos.

Por fim,o kernel Linux também busca inovações de desempenho e eficiência nas operações internas do sistema, como é o caso do arquivo io_uring.c, que introduz uma abordagem para I/O assíncrono.[6] Desde que foi incluído no kernel na versão 5.1, o io_uring tem passado por um grande volume de ajustes, devido à sua relevância em ambientes de alto desempenho. Da mesma forma, o arquivo Makefile centraliza o processo de compilação do kernel, com ajustes constantes para integrar novos módulos e otimizar a construção do sistema. A análise desses arquivos mostra a complexidade e o rigor no desenvolvimento do kernel, que não só busca estabilidade e compatibilidade, mas também inovação contínua para atender às demandas dos usuários e das empresas que utilizam o Linux.

4.2 Desenvolvedores e contribuições

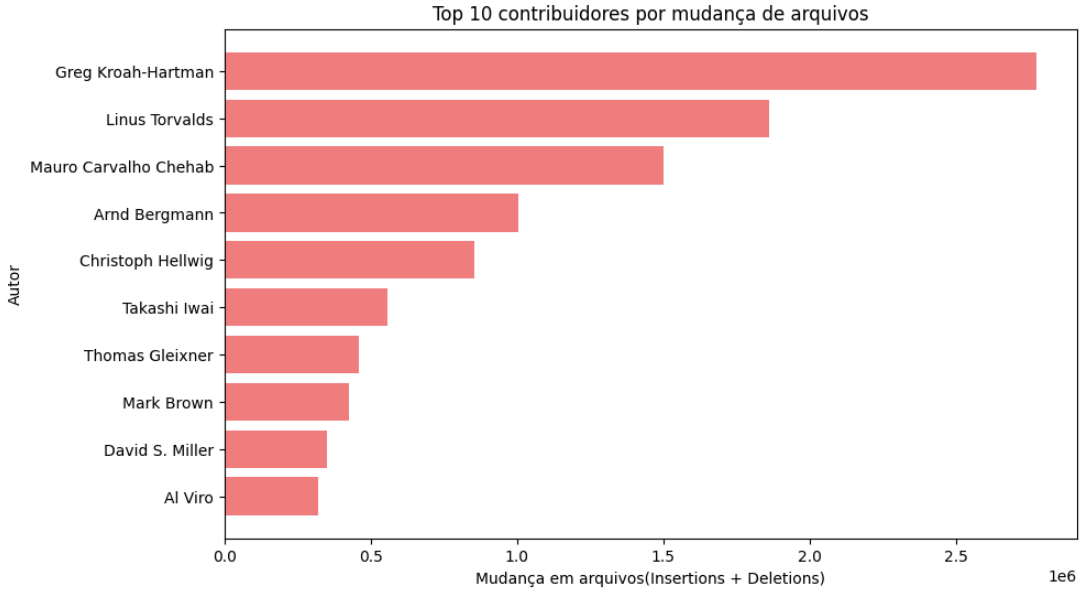
[6] io_uring introdução. Disponível em: https://kernel.dk/io_uring.pdf

Figura 2: contribuidores por números de commit



Fonte: Elaborado pelo Autor

Figura 3: contribuidores por mudança de arquivos

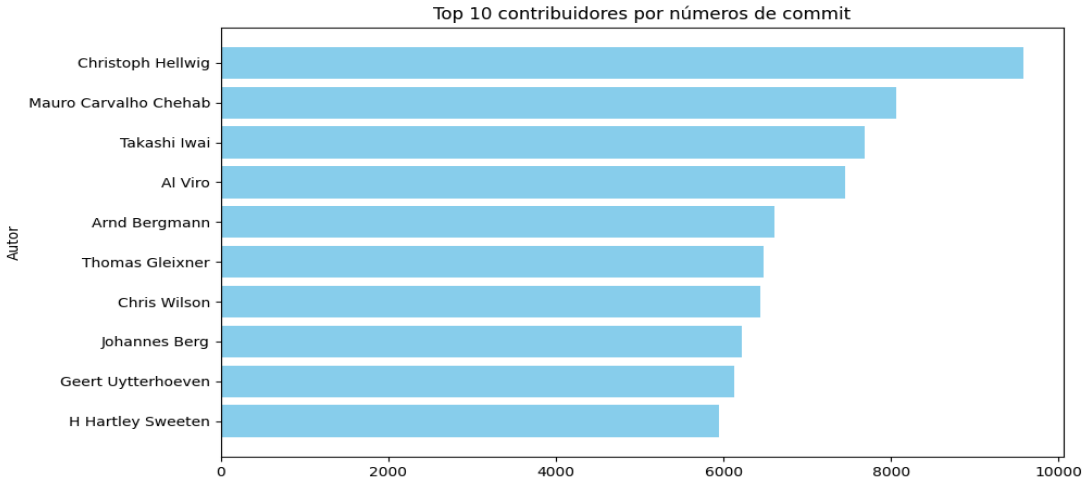


Fonte: Elaborado pelo Autor

Conforme vemos na figura 2 e 3 esses dois gráficos trazem toda a informação sobre o número de commits e mudanças de arquivos que passaram no projeto, porém isso também inclui um grande número de merge tags e merge branches, dito isto, existe uma necessidade de também extrair esses dados sem a inclusão desses valores que não alteram nada, mas servem como marcos e uma garantia de que nada mal-intencionado vá entrar no código. Essa separação foi necessária para evitar confusões durante o processo de levantar dados específicos sobre aspectos do kernel, tais como período de atividade relacionados a alterações

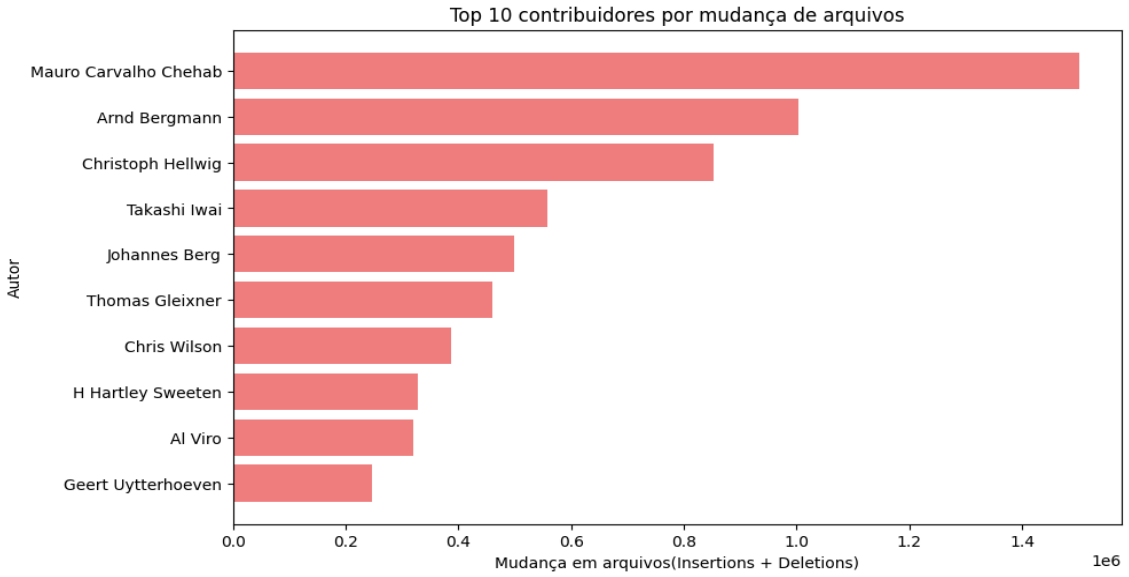
em período de tempo específicos ou frequência de commits em uma determinada data.

Figura 4: contribuidores por números de commit sem merge tags e branches



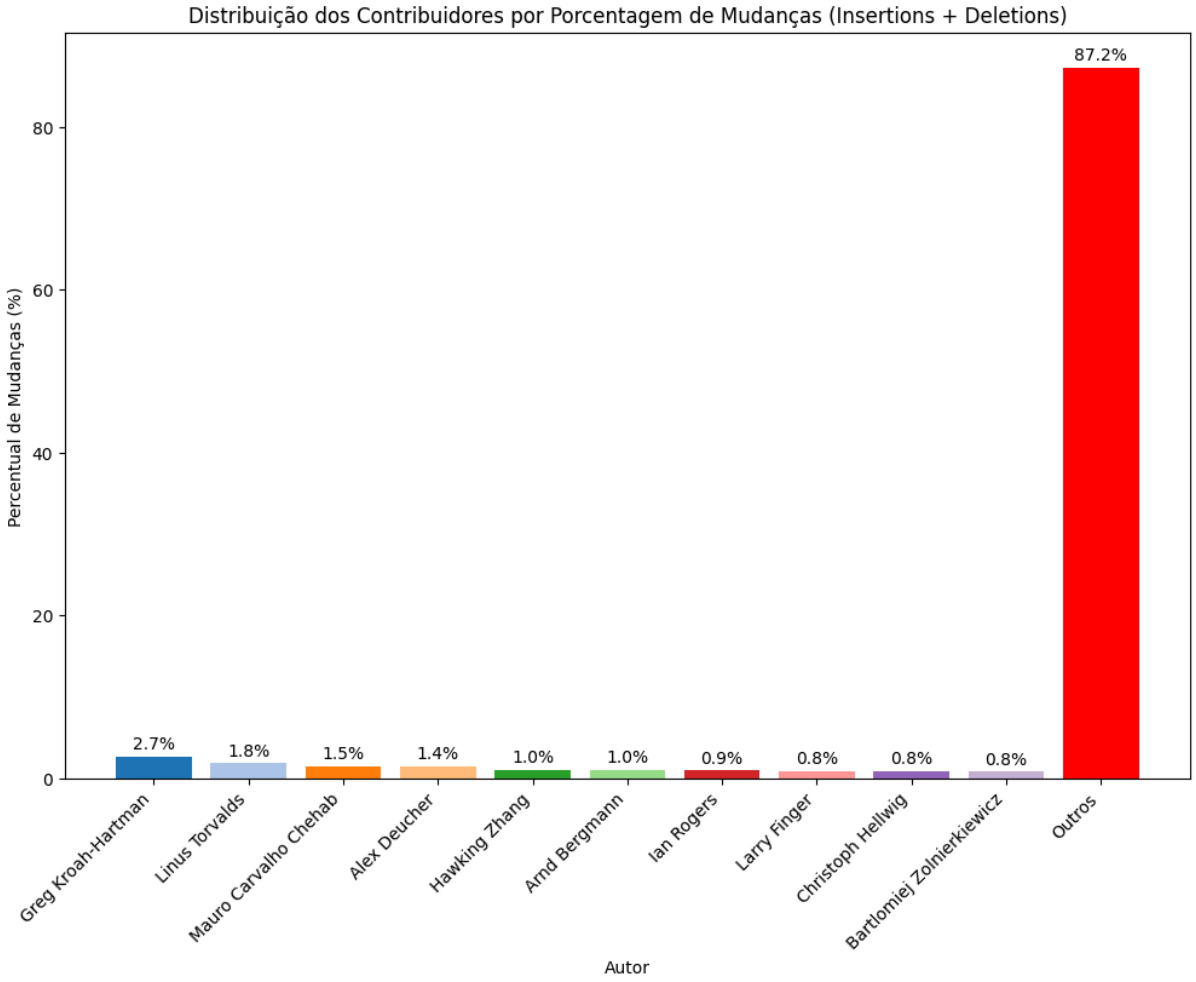
Fonte: Elaborado pelo Autor

Figura 5: contribuidores por mudança de arquivos sem merge tags e branches



Fonte: Elaborado pelo Autor

Figura 6: comparação em % com o top 10 e os demais contribuidores.



Fonte: Elaborado pelo Autor

Na figura 6, podemos ver que os contribuidores mais ativos, apesar de terem feito grandes contribuições, a comparação com o total de mudanças feitas por todos os contribuidores ainda é considerável, este gráfico também desconsidera merge branches e tags.

Tabela 3 - Contagem de commits por autor onde 'files_changed' é '[]'(merge tags):

Author	**Files Changed**
Linus Torvalds	38353
David S. Miller	11575
Mark Brown	5009
Arnd Bergmann	4649
Dave Airlie	3548

Fonte: Elaborado pelo Autor

Na tabela 3 temos os commits que não possuem conteúdo dentro das “files_changed” com seu autor e o total de commits por autor. além disso, o número total de autores que fizeram algum tipo de merge tag ou merge branch é de 7122, dos 34460 colaboradores totais.

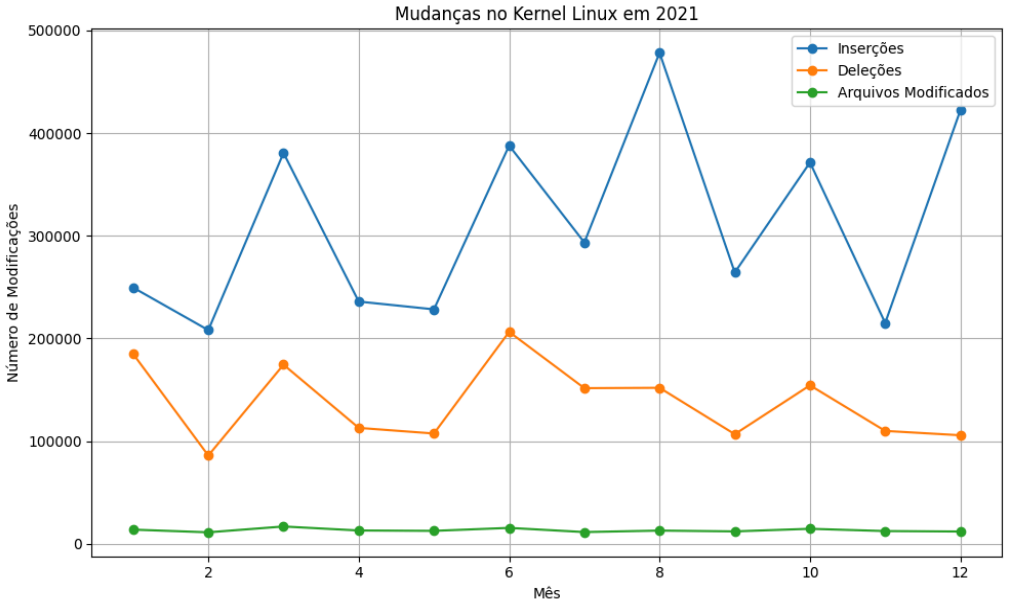
4.3 Criação do CSV(Comma-separated values)

O processo de criação do arquivo CSV exigiu um estudo aprofundado. Como todo o trabalho foi realizado em um ambiente Windows, foi necessário considerar possíveis problemas ao clonar o repositório do Linux. Um exemplo disso são os arquivos `aux.c` e `aux.h`, que, devido à sua função como componentes legados no Windows, possuem nomes reservados e não podem ser manipulados diretamente no sistema operacional. Para contornar essas limitações, foi necessário o uso do **Windows Subsystem for Linux (WSL)**, o que permitiu importar e manipular o projeto de forma eficiente.

4.4 Alterações perto de novas versões

Parte do processo de entender o fluxo de commit do Kernel, é saber como esse fluxo é tratado, neste trecho será apresentado o fluxo de mudança quando o linux está próximo a receber uma nova versão e meu entendimento sobre o processo, sendo esta a versão 5.15, que tratou principalmente da implementação do NTFS3, e correções de bugs.[7]

Figura 7: Análise de mudanças em 2021



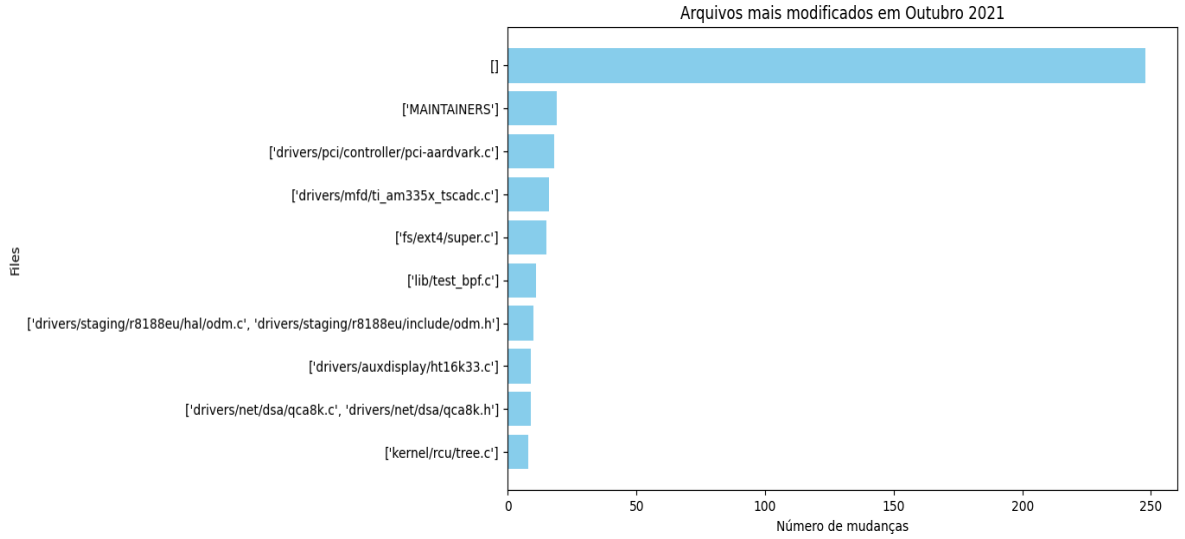
Fonte: Elaborado pelo Autor

[7] Linus Torvaldls,
Email referente a versão, disponível em:
https://lore.kernel.org/lkml/CAHk-=wjfbfQobW2jygMvgfJXKmzZNB=UTzBrFs2vTEzVpBXA4Q@mail.gmail.com/

Na figura 7, podemos ver o decorrer de inserções e deleções de 2021, a versão 5.15 teve sua release em outubro de 2021, com isso podemos ver superficialmente uma grande concentração de mudanças em agosto, março e dezembro, e é possível ver que isto segue um padrão, um mês tem um grande fluxo de commits, seguido de um com menos, sendo que não está considerando merge tags ou merge branches.

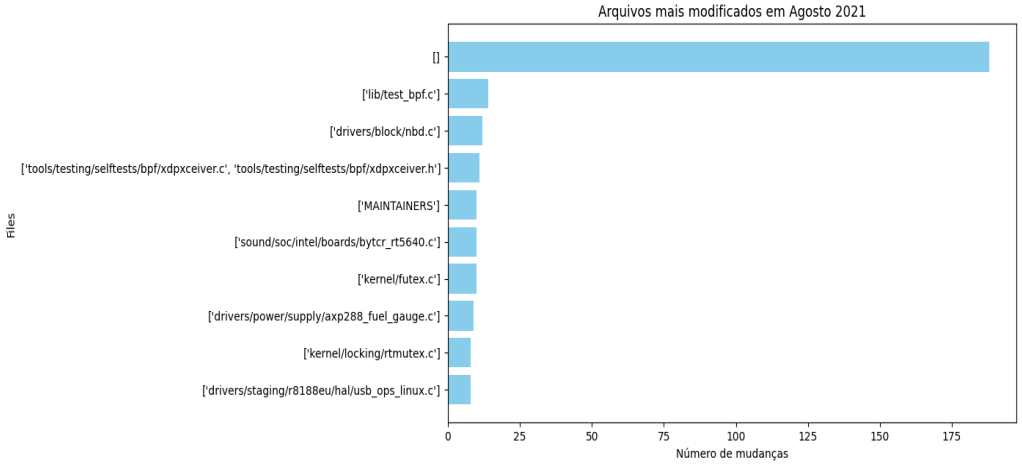
Vendo de uma forma superficial, dá pra ver um padrão de alterações ao decorrer do ano, na figura 8 e 9 podemos ver com mais detalhes o que exatamente está sendo alterado, uma grande concentração de mudança nos arquivos de drivers e testes, também podemos ver na figura 10 e 11 que, após a release, o número de merges diminui, tendo menos merge tags e um grande número de alterações diretamente no código, porém isso não altera o ritmo de commits como um todo.

Figura 8: Arquivos modificados outubro 2021



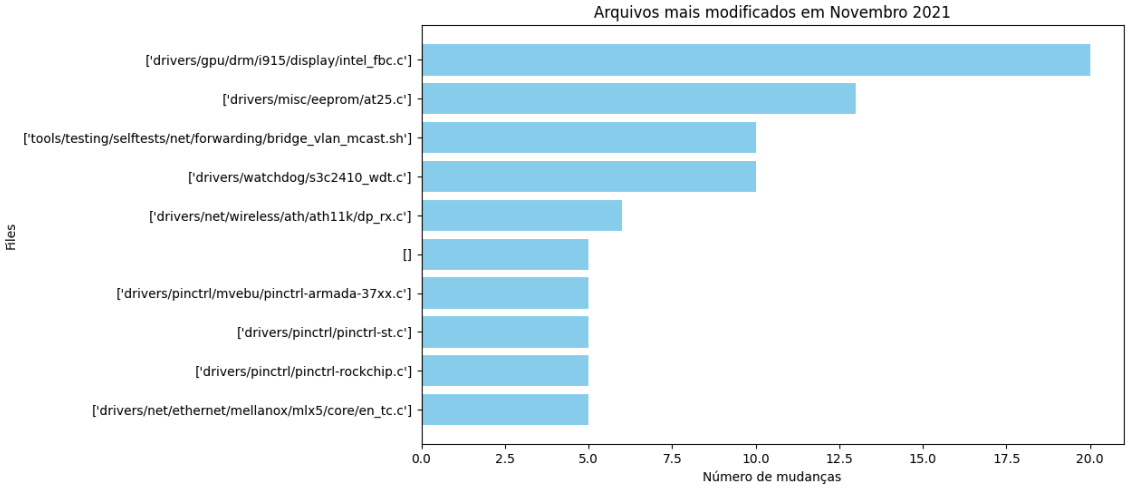
Fonte: Elaborado pelo Autor

Figura 9: Arquivos modificados agosto 2021



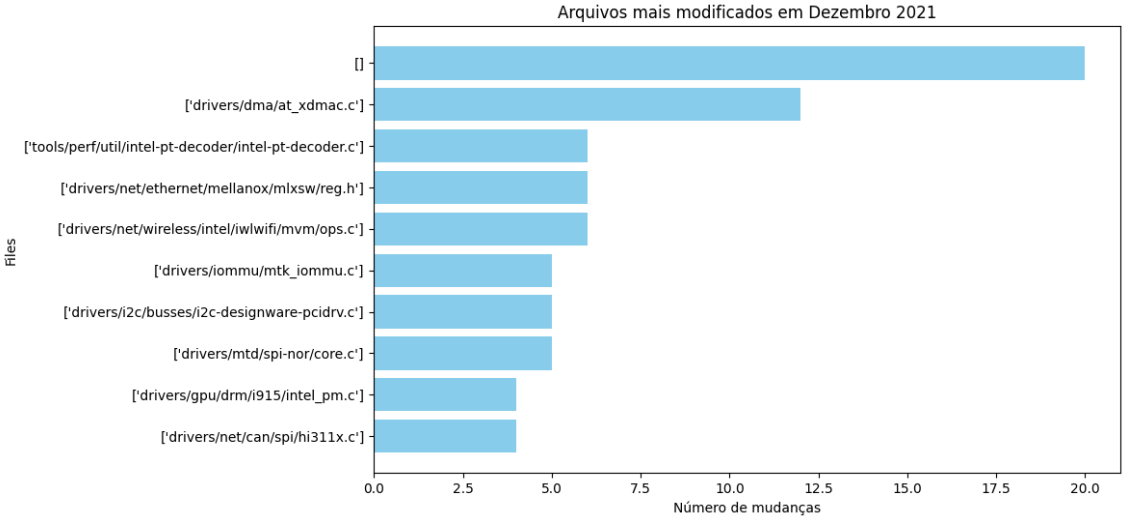
Fonte: Elaborado pelo Autor

Figura 10: Arquivos modificados novembro 2021



Fonte: Elaborado pelo Autor

Figura 10: Arquivos modificados novembro 2021



Fonte: Elaborado pelo Autor

5 Resultados

Os resultados da análise revelam algumas informações-chave sobre o desenvolvimento do kernel Linux:

- **Arquivo Mais Modificado**: O arquivo “MAINTAINERS” foi o mais alterado, seguido de arquivos relacionados a drivers indicando que o gerenciamento da equipe e atualizações relacionadas a compatibilidade são constantemente

monitorados e bem importantes para o projeto. Isso também reflete uma necessidade de manter a equipe de manutenção e de colaboradores organizada e atualizada, considerando a quantidade de pessoas que entram e saem do projeto, especialmente em um projeto de larga escala como o kernel Linux.

- **Foco em Drivers**: Os drivers, especialmente os relacionados à GPU da AMD,Intel, e drivers de som foram frequentemente modificados. Isso pode indicar tanto uma preocupação com a correção de bugs quanto a necessidade de suportar novos dispositivos e funcionalidades, o que mostra a atenção que empresas grandes dão para o projeto.
- **Contribuidores Mais Ativos**: A análise dos commits também permitiu identificar os desenvolvedores mais ativos e suas áreas de foco. Foi possível observar que certos desenvolvedores atuam mais em certos aspectos do código do que outros, também tendo uma variabilidade no decorrer de seu período de atividade, por exemplo Linus Torvalds, que até um certo período neste repositório era um desenvolvedor com mudanças bastante frequentes, e agora está mais focado em realizar a administração do que ativamente mudar o código em si.

6 Considerações Finais

Este projeto serviu como uma forma de aprendizado para ciência de dados e como uma oportunidade de aprofundar conhecimento sobre Linux e códigos open-source. Apesar das limitações de tempo e da minha familiaridade ainda inicial com ciência de dados, consegui desenvolver um trabalho que me trouxe satisfação e me ajudou a entender mais o desenvolvimento de software em larga escala.

Considerando a complexidade do kernel Linux, abordar muitas áreas ou peculiaridades poderia prejudicar o foco da análise. Dessa forma, decidi apresentar informações claras e úteis sobre os autores, suas participações e os arquivos mais modificados, o que contribui para destacar o valor desse tipo de pesquisa, mas também para abrir possibilidades para análises futuras que podem explorar períodos de tempo ou áreas mais específicas do código.

Este projeto não apenas apresenta uma análise, mas também detalha os processos e códigos utilizados, permitindo a aplicação de metodologias semelhantes em outros repositórios. Essa abordagem abre a possibilidade de compreender o fluxo de desenvolvimento de diferentes comunidades, ou até mesmo em ambientes profissionais fechados, ajudando a identificar aspectos relacionados aos desenvolvedores e ao fluxo do projeto como um todo. Com um entendimento mais profundo e informações suficientes, também se torna possível especular sobre possíveis alterações e versões futuras, além de identificar quais partes de um código são mais frequentemente modificadas e necessitam de maior manutenção. Embora a especulação não tenha sido o foco principal desta pesquisa, certos padrões e repetições puderam ser observados a partir dos dados apresentados.

Para trabalhos futuros, seria interessante procurar por lançamentos de drivers de grandes empresas, e como elas contribuem antes e depois ao kernel do linux, com a possibilidade de identificar quando elas estão contribuindo, graças ao formato de email padrão que cada uma trás, e também o que diferencia o mantenedor principal de cada versão, e o que trazem de diferença em relação a outros.

Ademais, os resultados demonstram a atenção dada em manter drivers e mantenedores atualizados para garantir a integridade do código. Até o final desta pesquisa, o número total de pessoas que realizaram merges no código-fonte foi de 7122, entre os 34460 desenvolvedores que já contribuíram de alguma forma.

Referências:

Documentação do Matplotlib. Disponível em: https://matplotlib.org/stable/. Acesso em: 10 nov. 2024.

Eficiência e introdução ao io_uring. Io_uring. Disponível em: https://kernel.dk/io_uring.pdf.

GODFREY, M. W.; TU, Q. Evolution in Open Source Software: A Case Study. Anais da Conferência Internacional sobre Manutenção de Software (ICMS), p. 131-142, 2000.

Histórico de versionamento. Disponível em: https://en.wikipedia.org/wiki/Linux_kernel_version_history. Acesso em: 15 out. 2024.

Jupyter. Try Jupyter Notebooks. Disponível em: https://jupyter.org/try-jupyter/notebooks/?path=notebooks/Intro.ipynb.

KOREN, O. A Study of the Linux Kernel Evolution. *Revista Internacional de Processos de Software de Código Aberto*, v. 2, n. 1, p. 49-67, 2006.

PALIX, N.; THOMAS, G.; CALVÈS, C.; LAWALL, J.; MULLER, G. Faults in Linux: Ten Years Later. Anais da Décima Sexta Conferência Internacional sobre Suporte Arquitetural para Linguagens de Programação e Sistemas Operacionais (ASPLOS), p. 305-318, 2011.

TORVALDS, L. *Linux Kernel Source*. Repositório GitHub. Disponível em: https://github.com/torvalds/linux. Acesso em: 10 nov. 2024.

TSAI, C. et al. Evolution of a Specific Part of the Linux Kernel. *Jornal de Sistemas e Software*, v. 116, p. 1-12, 2016.

Tutoriais do DataCamp. Disponível em: https://www.datacamp.com. Acesso em: 10 nov. 2024.

Wikipedia. Git. Disponível em: https://pt.wikipedia.org/wiki/Git.

APÊNDICE

A. Pesquisa em Jupyter

Disponível em: https://colab.research.google.com/drive/13H7rR7t2qWqAcMCm1eGSCmupC_9kq0aN#scrollTo=CPrepz9r1kZf

B. Vídeo breve explicativo do processo

Disponível em: https://www.youtube.com/watch?v=dVgf_CvOrus.