Следы создателя: географический и структурный анализ кода Линуса Торвальдса в ядре Linux

technothecow, Quizert, LevAlimpiev

Национальный исследовательский университет

«Высшая школа экономики»

Февраль 2026

Аннотация

Ядро Linux — один из крупнейших проектов с открытым исходным кодом, объединяющий тысячи разработчиков. Однако код его создателя, Линуса Торвальдса, по-прежнему присутствует в кодовой базе и эволюционирует вместе с ней. В данной работе мы анализируем не *объём*, а *характер и локализацию* кода Торвальдса: в каких подсистемах он сосредоточен, на каких языках написан, как меняется его «география» от версии к версии. Анализ проводится по данным `git blame` для четырёх знаковых версий ядра (3.0, 4.15, 5.10, 6.18), охватывающих период с 2011 по 2025 год. Результаты показывают, что абсолютный след Торвальдса неуклонно сокращается (с 3,5 млн до 2,0 млн строк), однако его код остаётся стабильно распределён между подсистемами, а доля периферии (`drivers/`, `sound/`) составляет ∼55% во всех версиях.

1 Введение

Ядро Linux, первая версия которого была опубликована Линусом Торвальдсом в 1991 году, к настоящему моменту является одним из самых масштабных совместных программных проектов в мире. По данным Linux Foundation, в разработке ядра приняли участие более 20 000 разработчиков из более чем 1 700 компаний [1].

Традиционный подход к анализу авторства кода сосредоточен на количественных показателях: *сколько* строк написал тот или иной разработчик. Однако в проекте масштаба ядра Linux такой подход упускает важную качественную составляющую: *где* именно в архитектуре системы сосредоточен код автора, *какого* он типа (исходный код, заголовки, ассемблер, конфигурация), и как эта картина меняется во времени.

В данной работе мы смещаем фокус с количества на структуру и локализацию. Мы анализируем только те файлы ядра, в которых по данным `git blame` присутствуют строки авторства Торвальдса, и исследуем:

- в каких подсистемах ядра сосредоточен его код;
- на каких языках и в каких типах файлов он пишет;

- насколько концентрирован или распределён его вклад;
- как «след создателя» мигрирует между подсистемами от версии к версии.

2 Обоснование выбора версий

Для анализа выбраны четыре версии ядра, каждая из которых маркирует значимый этап в эволюции проекта (табл. 1).

Таблица 1: Выбранные версии ядра и обоснование

Версия	**Дата**	**Обоснование**
3.0	Июль 2011	Символический «перезапуск» нумерации после серии 2.6.x (2003–2011). Удобная отправная точка: ядро уже использует Git (с 2005 г.), сообщество выросло, а эффект начальной миграции на Git — когда Торвальдс формально являлся автором ∼80% коммитов — уже сгладился.
4.15	Январь 2018	Версия, вышедшая сразу после обнаружения уязвимостей Meltdown и Spectre. Масштабные патчи безопасности от Intel, Google, ARM и др. привели к всплеску активности сторонних контрибьюторов, особенно в подсистемах `arch/` и `kernel/`.
5.10	Декабрь 2020	Версия с долгосрочной поддержкой (LTS), выпущенная в разгар пандемии COVID-19. Одна из ключевых LTS-версий, широко используемая в промышленных системах (Android, облачные платформы).
6.18	Май 2025	Одна из последних стабильных версий на момент исследования. Позволяет оценить текущее состояние проекта; роль Торвальдса сместилась к управлению и архитектурным решениям.

Выбор именно версий, а не произвольных дат, обусловлен тем, что каждый тег версии в Git соответствует конкретному стабильному состоянию кодовой базы, пригодному для воспроизводимого анализа. Период между версиями неравномерен, но это сознательный выбор: нас интересуют не временны́е интервалы, а качественные изменения в проекте.

3 Методология

3.1 Сбор данных

Для каждой из четырёх версий из репозитория ядра Linux извлекались данные `git blame` с фильтрацией по автору `Linus Torvalds`. Для каждого файла, содержащего хотя бы одну строку авторства Торвальдса, фиксировались:

- путь к файлу (`file_path`);
- общее количество строк в файле (`total_lines`);

- количество строк авторства Торвальдса (`linus_lines`).

Результаты агрегированы в CSV-файл формата:

```
snapshot_date,file_path,linus_lines,total_lines
21-07-2011,kernel/sched/core.c,150,8400
21-07-2011,mm/memory.c,45,3200
...
```

Суммарный объём данных: 42 002 записи по 15 626 уникальным файлам в четырёх версиях.

Важное ограничение. В выборку включены *только файлы, содержащие код Торвальдса*. Это означает, что мы не можем корректно вычислить долю Торвальдса среди *всех* файлов ядра. Однако для целей нашего исследования это ограничение не является недостатком: нас интересует не абсолютная доля, а *структура и характер* его вклада — в каких подсистемах, файлах и на каких языках сосредоточен его код.

Второе ограничение: эффект миграции на Git. Ядро Linux перешло на систему контроля версий Git в 2005 году. При миграции большая часть кода была зафиксирована в первом коммите с Линусом Торвальдсом в качестве автора, даже если этот код был написан другими разработчиками в более ранних системах контроля версий (BitKeeper, CVS). Это означает, что часть кода, атрибутированного Торвальдсу по данным `git blame`, на самом деле не является его авторством. Для точного анализа *истинного* вклада Торвальдса необходим анализ диффов между версиями, что выходит за рамки данной работы.

3.2 Извлечение признаков

Из пути к файлу (`file_path`) извлекаются следующие признаки:

- **Подсистема** — директория верхнего уровня (`kernel/`, `mm/`, `fs/`, `drivers/`, `net/`, `arch/`, `include/` и др.).
- **Тип файла** — расширение: `.c` (исходный код), `.h` (заголовки), `.S` (ассемблер), `Kconfig` (конфигурация), `Makefile` и др.
- **Глубина пути** — количество компонентов в пути (например, `kernel/sched/core.c` имеет глубину 3).

3.3 Метрики

На основе собранных данных вычисляются следующие группы метрик.

3.3.1 Метрики масштаба присутствия

- $N_{\text{files}}(v)$ — количество файлов с кодом Торвальдса в версии v;
- $L_{\text{total}}(v) = \sum_f \text{linus\_lines}(f, v)$ — суммарное число строк Торвальдса;
- $N_{\text{subsystems}}(v)$ — количество подсистем (директорий верхнего уровня), в которых присутствует его код.

3.3.2 Метрики концентрации

- **Доля в файле**: $p(f, v) = \text{linus\_lines}(f, v)/\text{total\_lines}(f, v)$ — для каждого файла;
- **Индекс концентрации Герфиндаля–Хиршмана (HHI)** по подсистемам:

$$\text{HHI}(v) = \sum_{s} \left(\frac{L_s(v)}{L_{\text{total}}(v)} \right)^2,$$

где $L_s(v)$ — число строк Торвальдса в подсистеме s. HHI близкий к 1 означает, что код сконцентрирован в одной подсистеме; низкий HHI — что код распределён равномерно.

3.3.3 Метрики распределения по типам

- Доля строк в файлах `.c`, `.h`, `.S`, и прочих — по каждой версии.

3.3.4 Структурные метрики

- **«Ядро» vs «периферия»**: доля строк в ключевых подсистемах (`kernel/`, `mm/`, `fs/`, `init/`, `ipc/`, `lib/`, `block/`) против периферийных (`drivers/`, `sound/`, `samples/`, `tools/`).

4 Результаты

4.1 Масштаб присутствия

Таблица 2: Масштаб присутствия кода Торвальдса по версиям

	3.0	**4.15**	**5.10**	**6.18**
Файлов с кодом Торвальдса	11 778	10 850	9 896	9 478
Суммарно строк Торвальдса	3 519 576	2 733 204	2 319 651	1 956 144
Подсистем с его кодом	19	20	21	23
Среднее строк/файл	298,8	251,9	234,4	206,4
Медиана строк/файл	115	88	75	57

Абсолютный «след» Торвальдса монотонно сокращается: количество файлов упало на 19,5% (с 11 778 до 9 478), суммарное число его строк — на 44,4% (с 3,52 млн до 1,96 млн). При этом число подсистем, в которых присутствует его код, выросло с 19 до 23 — его «географический охват» расширился, несмотря на сокращение объёма.

4.2 География кода: распределение по подсистемам

В таблице 3 представлены десять подсистем с наибольшим числом строк Торвальдса для каждой версии.

Таблица 3: Топ-10 подсистем по числу строк Торвальдса

Версия 3.0			**Версия 4.15**		
#	Подсистема	Строки	#	Подсистема	Строки
1	drivers	1 716 118	1	drivers	1 333 912
2	arch	737 029	2	arch	581 634
3	fs	346 212	3	fs	264 357
4	sound	241 896	4	sound	188 729
5	net	221 787	5	net	158 571
6	include	152 201	6	include	97 176
7	scripts	21 363	7	lib	30 588
8	kernel	17 841	8	scripts	19 116
9	mm	15 601	9	kernel	13 081
10	security	12 018	10	mm	10 902

Версия 5.10			**Версия 6.18**		
#	Подсистема	Строки	#	Подсистема	Строки
1	drivers	1 101 655	1	drivers	918 467
2	arch	462 533	2	arch	389 253
3	fs	245 478	3	fs	191 310
4	sound	180 077	4	sound	165 340
5	net	147 856	5	net	129 973
6	include	87 581	6	include	78 928
7	lib	29 297	7	lib	29 191
8	kernel	11 419	8	kernel	9 327
9	scripts	11 104	9	security	8 479
10	mm	10 191	10	scripts	8 209

Во всех четырёх версиях лидером остаётся `drivers/`, за ним — `arch/`, `fs/`, `sound/` и `net/`. Порядок первых шести позиций не менялся ни разу; сдвиги наблюдаются только во второй половине рейтинга.

4.3 Типы файлов: на чём пишет Торвальдс

Таблица 4: Распределение строк Торвальдса по типам файлов

Тип	**3.0**	**4.15**	**5.10**	**6.18**
`.c` (исходный код)	2 608 391	2 021 948	1 726 922	1 432 135
`.h` (заголовки)	593 158	435 844	359 084	303 657
`.S` (ассемблер)	185 951	161 537	142 246	132 338
Прочие	93 633	85 450	69 463	70 186
`Kconfig`	28 678	20 872	15 920	12 889
`Makefile`	9 765	7 553	6 016	4 939

Подавляющее большинство строк Торвальдса — это исходный код на C (`.c`), составляющий $\sim$73% его вклада во всех версиях. На втором месте — заголовки (`.h`), далее —

ассемблер (`.S`). Конфигурационные файлы (`Kconfig`, `Makefile`) занимают менее 2% суммарно.

4.4 Концентрация: индекс HHI по подсистемам

Таблица 5: Индекс концентрации Герфиндаля–Хиршмана по подсистемам

Версия	HHI	Строки Линуса	Подсистемы
3.0	0,302	3 519 576	19
4.15	0,302	2 733 204	20
5.10	0,288	2 319 651	21
6.18	0,283	1 956 144	23

HHI остаётся стабильным — около 0,3 во всех версиях, с незначительным снижением от 3.0 к 6.18. Это означает, что распределение кода Торвальдса по подсистемам *не меняется принципиально*, несмотря на сокращение абсолютного объёма и рост числа подсистем.

4.5 Ядро vs периферия

Таблица 6: Строки Торвальдса: ключевые подсистемы vs периферия

Версия	Группа	Строк	Файлов	Доля строк
3.0	core	400 045	1 103	11,4%
3.0	periphery	1 958 015	3 998	55,6%
3.0	other	1 161 516	6 677	33,0%
4.15	core	329 192	1 184	12,0%
4.15	periphery	1 522 672	3 863	55,7%
4.15	other	881 340	5 803	32,2%
5.10	core	304 487	1 173	13,1%
5.10	periphery	1 281 770	3 548	55,3%
5.10	other	733 394	5 175	31,6%
6.18	core	243 753	1 124	12,5%
6.18	periphery	1 083 923	3 691	55,4%
6.18	other	628 468	4 663	32,1%

Доля периферии (`drivers/`, `sound/`, `tools/`) в строках Торвальдса устойчиво составляет ∼55% во всех версиях. Ядро (`kernel/`, `mm/`, `fs/`, `init/`, `ipc/`, `lib/`, `block/`) занимает ∼11–13%.

Этот результат *противоречит* наивному ожиданию, что основатель концентрируется на «ядре ядра». Объяснение: подсистема `drivers/` — самая крупная по объёму часть Linux, и Торвальдс исторически являлся автором значительной части раннего кода драйверов, который постепенно вытесняется, но всё ещё присутствует.

4.6 Ключевые наблюдения

1. **Устойчивое сокращение.** Абсолютное число строк Торвальдса падает монотонно: -44% за период 3.0–6.18. Число файлов — аналогично (-20%). При этом среднее число строк на файл тоже снижается (с 299 до 206), что говорит об «размывании» его вклада внутри файлов.

2. **Рост географии.** Число подсистем, в которых присутствует код Торвальдса, выросло с 19 до 23, включая появление `rust/` в версии 6.18.

3. **Стабильная структура.** Ранжирование подсистем по объёму кода Торвальдса практически не меняется: `drivers` > `arch` > `fs` > `sound` > `net` > `include` во всех четырёх версиях. Индекс HHI стабилен ($\approx 0{,}3$).

4. **Преобладание C.** Торвальдс — автор преимущественно C-кода (`.c` $\approx 73\%$) и заголовков (`.h` $\approx 16\%$). Доля ассемблера составляет $\sim 5\%$, остальные типы — менее 3%.

5. **Периферия > ядро.** Вопреки ожиданиям, большая часть оставшегося кода Торвальдса находится в периферии ($\sim 55\%$), а не в ключевых подсистемах ядра ($\sim 12\%$).

5 Обсуждение

Полученные результаты позволяют составить «портрет» вклада основателя в крупном проекте с открытым исходным кодом.

Главный вывод — *форма* вклада Торвальдса удивительно стабильна при драматическом сокращении его *объёма*. Структура распределения по подсистемам, типам файлов и соотношение «ядро/периферия» практически не изменились за 14 лет (3.0–6.18), несмотря на то что абсолютное число его строк сократилось почти вдвое.

Это наблюдение согласуется с гипотезой о *пропорциональном вытеснении*: код Торвальдса замещается другими контрибьюторами примерно равномерно во всех подсистемах, а не вытесняется из одних подсистем в пользу других.

Неожиданным является высокая доля периферии ($\sim 55\%$) в его коде. Объяснение кроется в том, что `drivers/` — исторически самая объёмная часть ядра, и Торвальдс являлся автором значительной доли раннего кода драйверов. Хотя этот код постепенно переписывается, он всё ещё составляет наибольшую абсолютную долю.

С точки зрения программной инженерии, подобный паттерн — когда основатель переходит от широкого написания кода к точечным вмешательствам, а его исторический вклад «размывается» равномерно — можно рассматривать как естественную стратегию масштабирования open-source проекта.

6 Заключение

В данной работе мы провели структурный анализ кода Линуса Торвальдса в ядре Linux на протяжении четырёх ключевых версий (3.0, 4.15, 5.10, 6.18). В отличие от традиционного количественного подхода, мы сосредоточились на *географии* и *характере* кода: его распределении по подсистемам, типам файлов и структуре «ядро vs периферия».

Основные результаты:

- Абсолютный след Торвальдса сокращается (-44% строк за 14 лет), но его структурное распределение остаётся стабильным (HHI $\approx 0{,}3$).
- Код преимущественно на C и в заголовках; доминирующие подсистемы — `drivers/`, `arch/`, `fs/`.
- Периферия устойчиво составляет $\sim 55\%$ его вклада.

Предложенная методология и набор метрик могут быть применены для анализа роли основателей в других крупных open-source проектах.

Направления для дальнейших исследований. Важно отметить, что данные `git blame` несут искажение, связанное с миграцией ядра на Git в 2005 году: значительная часть кода, атрибутированного Торвальдсу, могла быть написана другими разработчиками в более ранних системах контроля версий. Для точного анализа *истинного* вклада необходимо исследование диффов между версиями (а не кумулятивного состояния), что позволит отделить код, написанный Торвальдсом в рамках Git-истории, от кода, унаследованного при миграции. Такое исследование может стать ценным дополнением к настоящей работе.

Список литературы

[1] Linux Foundation, *2020 Linux Kernel History Report*, 2020. https://www.linuxfoundation.org/resources/publications/linux-kernel-history-report-2020

[2] Torvalds, L., *The Git Version Control System*, 2005. https://git-scm.com/

[3] Kroah-Hartman, G., Corbet, J., McPherson, A., *Linux Kernel Development: How Fast It is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*, Linux Foundation, 2006–2020.

[4] Raymond, E. S., *The Cathedral and the Bazaar*, O'Reilly Media, 1999.

[5] Mockus, A., Fielding, R. T., Herbsleb, J. D., *Two Case Studies of Open Source Software Development: Apache and Mozilla*, ACM Transactions on Software Engineering and Methodology, 11(3), 2002.