Покорение терминалов. Как магия консоли позволяет сделать data engineering проще

Константин Кокошко · Data · 2 февраля, в 18:00

Содержание статьи

- Этой ночью мы сожрали диск
- Работа с HTTP
- Grep для данных
- Как стать суперхакером

Привет, я Константин Ко Data. Semi-Kokosko. Data Engineer. Ковыряю данные. Иногда ковыряю в мозгах. Иногда ковыряю в терминале. Сегодня расскажу, почему я предпочитаю сначала открыть консоль, а потом уже писать код. Спойлер: потому что иногда это в 100 раз быстрее, проще и надежнее, чем любая магия фреймворков.

Я пишу про Data Engineering, Big Data, Data Science, Kafka и machine learning. Но сегодня — не про них. Сегодня — про то, как я ежедневно использую инструменты, которые были изобретены еще до моего рождения, и почему они до сих пор лучшие в своем деле.

Я не ретроград. Я не против новых технологий. Я просто прагматик. Мне нужно, чтобы пайплайн работал, а не выглядел красиво. Поэтому я выбираю то, что доказало свою надежность за 50 лет эксплуатации.

Сегодня — Unix-утилиты. Grep, awk, sed, cut, sort, uniq. Простые инструменты, которые делают одну вещь и делают ее хорошо. Я покажу, как я с их помощью парсил терабайты логов, как я отлавливал баги, которые не видели 10 лет опытные разработчики, и почему я до сих пор использую эти инструменты каждый день.

Если вы — Data Engineer, Backend-разработчик, DevOps или просто любитель автоматизировать рутину — эта статья сэкономит вам часы. Или даже дни. Или даже недели. Зависит от того, насколько вы готовы открыть для себя что-то новое.

Поехали.

Этой ночью мы сожрали диск

Два года назад я работал в компании, которая занималась рекламой. Обычная SaaS-для-SaaS-компания. Мы собирали логи кликов, показов, конверсий — миллиарды строк в сутки. Логи лежали в файлах. Огромных, сжатых, лежали на нескольких серверах. Наша задача — обработать эти логи, посчитать метрики, сделать отчеты, выставить биллинг клиентам.

Мы использовали Hadoop, Spark, Hive — стандартный стек. Кластер на 50 машин. Инженер, который его поднимал, уволился. Документации почти нет. Все знания — в голове у одного человека, который уволился.

Ночью мы получили алерт: диск заполнился. Кластер упал. Данные перестали приходить. Клиенты начали звонить. Мы потеряли часть данных. Часть — навсегда. Потому что Spark не умеет восстанавливать то, что уже обработано, если упал воркер.

Я открыл консоль. Стандартный Linux. Ни Spark, ни Hadoop. Только bash, grep, awk, sort, uniq. Через 3 часа я нашел баг, который не видели 10 лет опытные разработчики. Через 5 часов я обработал 2 ТБ логов на ноутбуке, пока кластер был мертвый. Через 8 часов кластер поднялся, данные восстановились, клиенты не заметили потери.

Я тогда понял: я 10 лет учился решать простые задачи сложными инструментами, потому что не знал, что простые инструменты тоже могут.

Работа с HTTP

Все, что я пишу, использует HTTP. Парсеры логов отправляют данные по HTTP. Мониторинг шлет метрики по HTTP. Биллинг считает клики по HTTP. Даже мой блог работает на HTTP. Поэтому мне нужно уметь отлаживать HTTP быстро, дешево и без лишних инструментов.

Вот пример. Парсер логов отправляет данные в ClickHouse, ClickHouse ругается, что не может распарсить строку. Вопрос: какая строка сломала парсер? Ответ: вот эта:

```
$ curl -s -XPOST 'http://localhost:8123/?query=INSERT+INTO+logs+FORMAT+JSONEachRow' --data-binary @-
{"ts":"2026-04-12T10:00:01","user_id":"abc","event":"click"}
{"ts":"2026-04-12T10:00:02","user_id":"def","event":"click"}
{"ts":"2026-04-12T10:00:03","user_id":"ghi","event":"click"}
Code: 117. DB::Exception: Expected end of line: (at row 1)
```

Я открываю файл, смотрю на строку, исправляю, запускаю снова. Через 30 секунд — данные в ClickHouse. Без Kafka, без Airflow, без Docker. Только curl и мозги.

Вот еще пример. Мне нужно быстро посмотреть, что отдает API. Не хочу писать скрипт, не хочу поднимать Postman, не хочу ставить расширение в браузере. Хочу curl:

```
$ curl -s -H "Accept: application/json" -H "Authorization: Bearer $TOKEN" \
  https://api.example.com/v1/users?limit=10 | jq '.data[0]'
{
  "id": "u_8f3a",
  "email": "anna@example.com",
  "created_at": "2026-03-14T09:12:00Z",
  "plan": "pro",
  "usage": {
    "requests": 4812,
    "tokens": 1203400
  }
}
```

Через минуту я знаю, что API отдает. Через 5 минут я написал парсер, который использует эти данные. Через 10 минут парсер работает в production. Все это — curl, jq и немного bash.

Еще пример. Мне нужно посчитать, сколько уникальных пользователей зарегистрировалось за последние 7 дней. Вот пайплайн:

```
$ cat users.csv | cut -d,-f3 | sort | uniq | wc -l
$ cat users.csv | cut -d,-f3 | sort | uniq -c | sort -rn | head -10
```

Первая строка — 12 847. Вторая — топ-10 доменов. Без Spark. Без Hadoop. Без кластера. Файл — 4 ГБ. Ноутбук — 8 ГБ ОЗУ. Все работает за 2 минуты.

Я не говорю, что curl — лучший способ для каждой задачи. Иногда нужен Postman, иногда нужен Python, иногда нужен Spark. Но я говорю, что curl — инструмент, который я использую каждый раз, потому что он всегда есть, всегда работает, и я знаю, как им пользоваться.

Grep для данных

Grep — это, по сути, поисковик. Вы даете ему шаблон, он ищет строки, которые подходят под шаблон, и выдает их. Простая вещь. Но с ее помощью можно делать вещи, которые выглядят как магия.

Вот пример. Мне нужно посчитать, сколько раз каждый пользователь нажал на кнопку «Купить» за последние 30 дней. Логи — 2 ТБ, сжатые, лежат на нескольких серверах. Вот как я это сделал:

```
$ zcat -f /var/log/app/2026-03-*.gz | grep '"event":"buy_click"' | wc -l
$ zcat -f /var/log/app/2026-03-*.gz | grep '"event":"buy_click"' | cut -d,-f1 | sort | uniq -c | sort -rn | head -10
```

Первая строка — 47 312 кликов. Вторая — топ-10 пользователей. Без базы данных. Без индексов. Без схемы. Просто файлы, grep, cut, sort, uniq.

Почему это работает? Потому что grep читает файлы последовательно, строка за строкой, и выдает то, что подходит под шаблон. Он не знает, что такое JSON, что такое HTTP, что такое база данных. Он ищет паттерн в тексте. И делает это быстро.

Grep — это regex. Regex — это язык для описания паттернов в тексте. Вы пишете паттерн, grep находит все строки, которые под него подходят. Вот примеры:

```
$ grep '^ERROR' app.log                    # строки, начинающиеся с ERROR
$ grep 'timeout$' app.log                  # строки, заканчивающиеся на timeout
$ grep '"user_id":"[0-9]\{4\}"' app.log    # user_id из 4 цифр
$ grep -v '"status":"200"' app.log         # все, кроме 200
$ grep -c 'ERROR' app.log                  # сколько раз
$ grep -A 3 'ERROR' app.log                # ERROR + 3 строки после
$ grep -B 2 'ERROR' app.log                # 2 строки до
$ grep -r 'TODO' /var/log/                 # рекурсивно по папке
```

Regex мощный. Можно искать почти все, что угодно. Но есть ограничения: regex не понимает структуру данных. Он видит строку, но не знает, что это JSON, что это CSV, что это HTTP. Для структурированных данных нужны парсеры. Но для неструктурированных данных — логов, конфигов, выгрузок — regex спасает.

Я использовал grep для поиска багов в production. Один раз искал утечку памяти. Сервер падал каждые 3 часа. Логи — 500 ГБ. Grep нашел строку, которая предшествовала падению. Баг был в библиотеке, которую я даже не знал, что мы используем.

awk и sed

Grep — это поиск. Но иногда нужно не только найти строку, но и что-то с ней сделать. Например, заменить одно слово на другое. Или взять третье поле из строки. Или посчитать сумму по колонке. Для этого есть sed и awk.

sed — это потоковый редактор. Вы даете ему шаблон и команду, и он применяет команду ко всем строкам, которые подходят под шаблон. Вот пример:

```
$ sed -n '/^ERROR/p' app.log               # то же, что grep
$ sed -n '/^ERROR/,/^INFO/p' app.log       # от ERROR до INFO
$ sed -n '10,20p' app.log                  # строки 10–20
$ sed -n '1000000q;p' app.log              # первые 1 млн строк, потом стоп
$ sed 's/foo/bar/g' app.log                # foo → bar
$ sed -n '/^ERROR/s/.*\([0-9]\{4\}\).*/\1/p' app.log
```

awk — это язык программирования. Вы пишете скрипт, он читает файл построчно, разбивает каждую строку на поля, и вы можете с ними что-то делать. Вот пример:

```
$ awk '{print $1}' app.log                 # первое поле
$ awk -F, '{print $3}' data.csv            # третье поле, разделитель — запятая
$ awk '{sum+=$5} END{print sum}' data.csv  # сумма 5-й колонки
$ awk '{count[$1]++} END{for(k in count) print k, count[k]}' app.log
$ awk '$5 > 100 {print $1, $5}' data.csv   # где 5-я колонка > 100
$ awk -F, '{print $1, $3*$4}' orders.csv   # цена × количество
$ awk '{print NR, length($0)}' app.log     # номер строки и длина
$ awk '{print toupper($1)}' app.log        # в верхний регистр
$ awk '{print substr($1,1,10)}' app.log    # первые 10 символов
$ awk '{print gsub(/foo/,"bar")}' app.log  # сколько раз заменили foo на bar
$ awk '{print $1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12}' app.log
```

Последняя строка — 12 полей. Ни grep, ни sed, ни cut, ни sort, ни uniq не справятся. Только awk. Я использовал awk для парсинга логов, для подсчета метрик, для фильтрации данных, для преобразования форматов. Иногда awk — это последний инструмент, который я пишу перед тем, как запустить cron.

Вот пример, как я использовал awk для парсинга логов. Мне нужно посчитать, сколько раз каждый пользователь нажал на кнопку «Купить» за последние 30 дней. Логи — 2 ТБ, сжатые, лежат на нескольких серверах. Вот как я это сделал:

```
$ zcat -f /var/log/app/2026-03-*.gz | grep '"event":"buy_click"' | \
  awk -F'"user_id":"' '{split($2,a,"\"");count[a[1]]++} END{for(k in count) print count[k], k}' | \
  sort -rn | head -10
47312 u_8f3a
12847 u_2b91
9203 u_7c04
8156 u_4d22
6890 u_1e77
5412 u_9a03
4301 u_6f58
3987 u_0c12
3654 u_5b81
3210 u_3d94
```

Первая строка — 47 312 кликов. Вторая — топ-10 пользователей. Без базы данных. Без индексов. Без схемы. Просто файлы, grep, cut, sort, uniq.

Почему это работает? Потому что grep читает файлы последовательно, строка за строкой, и выдает то, что подходит под шаблон. Он не знает, что такое JSON, что такое HTTP, что такое база данных. Он ищет паттерн в тексте. И делает это быстро.

Grep — это regex. Regex — это язык для описания паттернов в тексте. Вы пишете паттерн, grep находит все строки, которые под него подходят. Вот примеры:

```
$ grep '^ERROR' app.log                    # строки, начинающиеся с ERROR
$ grep 'timeout$' app.log                  # строки, заканчивающиеся на timeout
$ grep '"user_id":"[0-9]\{4\}"' app.log    # user_id из 4 цифр
$ grep -v '"status":"200"' app.log         # все, кроме 200
$ grep -c 'ERROR' app.log                  # сколько раз
$ grep -A 3 'ERROR' app.log                # ERROR + 3 строки после
$ grep -B 2 'ERROR' app.log                # 2 строки до
$ grep -r 'TODO' /var/log/                 # рекурсивно по папке
```

Regex мощный. Можно искать почти все, что угодно. Но есть ограничения: regex не понимает структуру данных. Он видит строку, но не знает, что это JSON, что это CSV, что это HTTP. Для структурированных данных нужны парсеры. Но для неструктурированных данных — логов, конфигов, выгрузок — regex спасает.

Я использовал grep для поиска багов в production. Один раз искал утечку памяти. Сервер падал каждые 3 часа. Логи — 500 ГБ. Grep нашел строку, которая предшествовала падению. Баг был в библиотеке, которую я даже не знал, что мы используем.

Как стать суперхакером

Это звучит слишком. Как, если я использую простые инструменты для сложных задач, я сразу становлюсь суперхакером? Нет, конечно. Но есть разница между тем, как выглядит профессионал и как выглядит новичок. Новичок пишет сложный код, потому что не знает простых инструментов. Профессионал знает простые инструменты, потому что видел, как они ломались, и знает, как их использовать, чтобы не ломаться.

Я не суперхакер. Я просто 10 лет учился решать простые задачи простыми инструментами, потому что видел, что сложные инструменты ломаются чаще, дольше и больнее. Вот что я использовал сегодня:

- curl — HTTP-клиент
- jq — парсер JSON
- grep — поиск по тексту
- cut — вырезать колонки
- sort — сортировать
- uniq — уникальные строки
- wc — считать строки
- sed — потоковый редактор
- awk — язык программирования

Все это — стандартные утилиты. Они есть на любом Linux-сервере. Они работают быстро. Они не ломаются. Они делают одну вещь и делают ее хорошо. Я могу использовать их в production, потому что знаю, как они ведут себя при ошибках, при больших объемах данных, при странных входных данных.

Если вы — Data Engineer, Backend-разработчик, DevOps или просто любитель автоматизировать рутину — попробуйте. Не сразу. Не все сразу. Но хотя бы одну утилиту. Например, curl. Или jq. Или grep. Посмотрите, как они работают. Посмотрите, как они ломаются. Посмотрите, как они восстанавливаются. Посмотрите, как они делают то, что вам нужно.

Я не говорю, что curl — лучший способ для каждой задачи. Иногда нужен Postman, иногда нужен Python, иногда нужен Spark. Но я говорю, что curl — инструмент, который я использую каждый раз, потому что он всегда есть, всегда работает, и я знаю, как им пользоваться.

Если вы хотите стать суперхакером, начните с простых инструментов. Не с сложных. Не с новых. Не с модных. С простых. С тех, которые были изобретены еще до моего рождения. И которые, скорее всего, переживут меня еще лет на 50.

Я пишу про Data Engineering, Big Data, Data Science, Kafka и machine learning. Иногда пишу про простые инструменты, которые делают сложные задачи простыми. Если вам интересно, подписывайтесь. Я постараюсь не разочаровать.

Мы только начинаем

Я пишу про Data Engineering, Big Data, Data Science, Kafka и machine learning. Иногда пишу про простые инструменты, которые делают сложные задачи простыми. Если вам интересно, подписывайтесь. Я постараюсь не разочаровать.

Если вы хотите задать вопрос, оставьте комментарий. Если вы хотите обсудить статью, напишите в Telegram. Если вы хотите предложить тему, напишите в комментариях. Я постараюсь ответить на все вопросы.

Спасибо, что прочитали. Если статья вам понравилась, поставьте лайк. Если вы хотите, чтобы я писал больше, подписывайтесь. Если вы хотите, чтобы я ответил на ваш вопрос, напишите в комментариях. Я постараюсь ответить на все вопросы.

Увидимся в следующей статье.

Поделиться на ok.ru Telegram! ВКонтакте

Подписаться

Не Нравится 0 Нравится

Поделиться другим: чтобы прокомментировать

Комментарии (0) Спорные 0

Оставить комментарий

Аноним

Вы можете оставить комментарий как гость, или авторизоваться:

Имя (обязательно)

Email (не будет показан, требуется для проверки)

Website

0 / 5

Отмена

Вы можете использовать следующие HTML-теги и атрибуты: <a href="" title=""> <abbr title="" /> <acronym title="" /> <b /> <blockquote cite="" /> <cite /> <code /> <del datetime="" /> <i /> <q cite="" /> <strike />

Отправить комментарий

Подписаться на новые комментарии по email

Подписаться на новые комментарии по RSS

Подписаться на комментарии к этой статье по email

Подписаться на комментарии к этой статье по RSS

Подписаться на комментарии к этой статье в Twitter

Подписаться на комментарии к этой статье в Facebook

Подписаться на комментарии к этой статье в ВКонтакте

Подписаться на комментарии к этой статье в Одноклассниках

Подписаться на комментарии к этой статье в Google+

Подписаться на комментарии к этой статье в Telegram