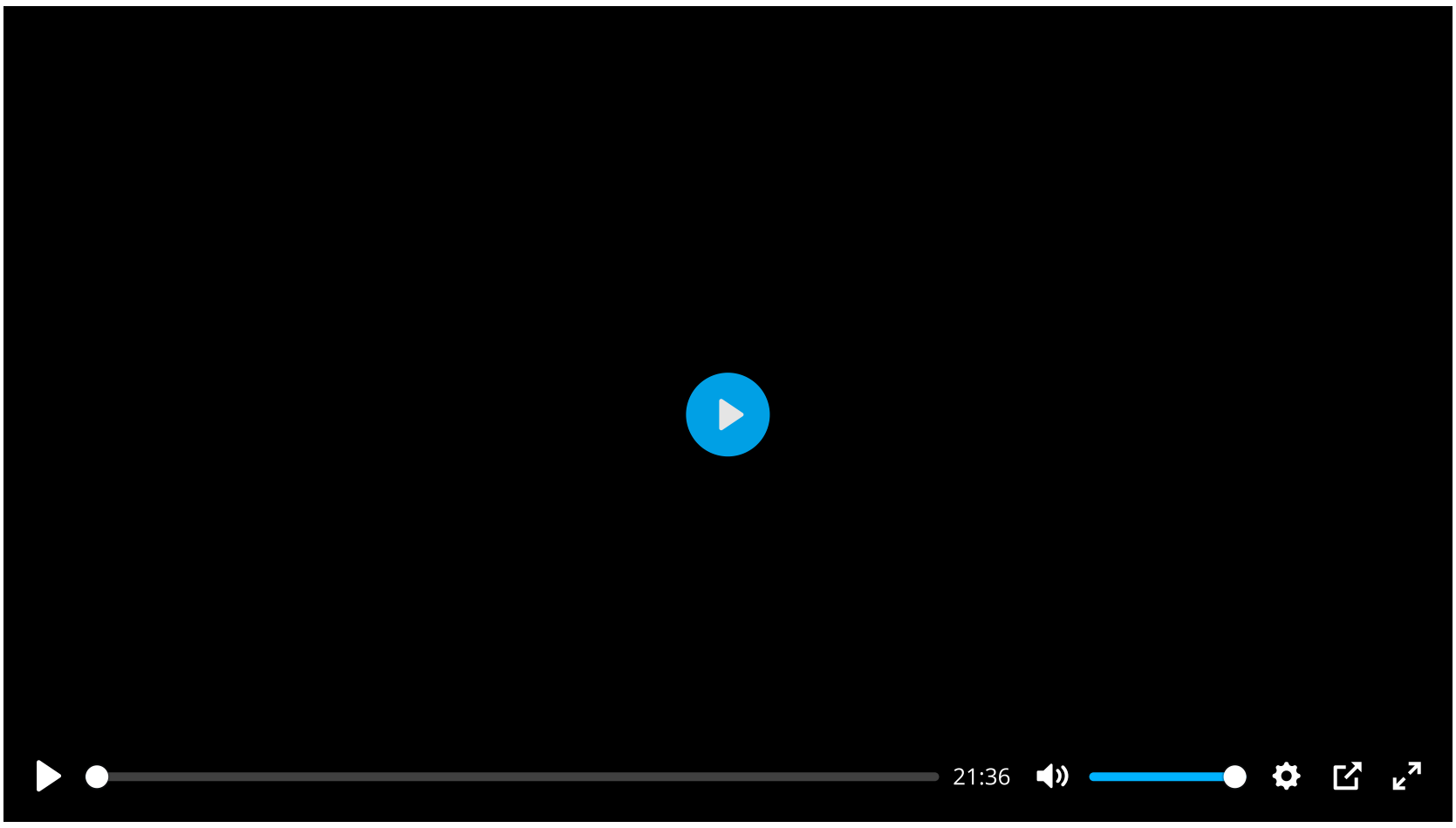
2.5 Linux терминал для задач DE



Текстовая расшифровка видео:

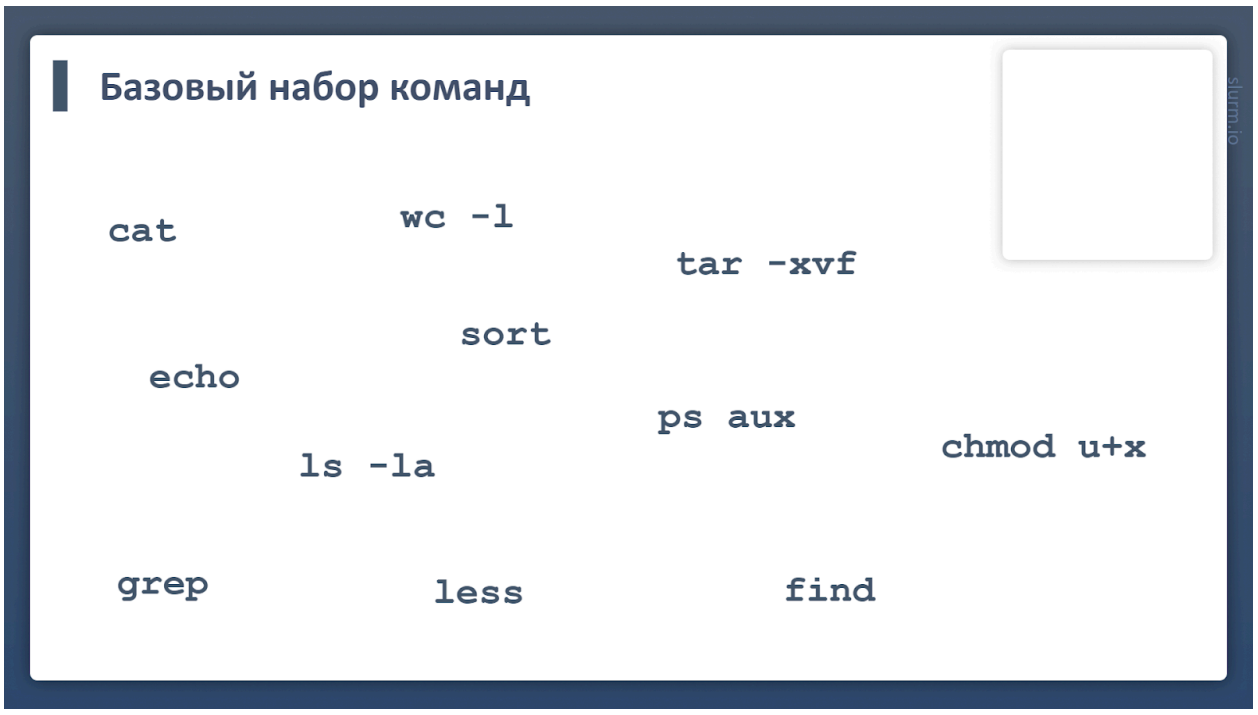
ИТАК, У МЕНЯ ЕСТЬ ТЕКСТОВЫЕ ДАННЫЕ...

План:

- Базовый набор команд;
- Первые шаги;
- Трубоукладка.

Базовый набор команд

Обратите внимание на список команд:

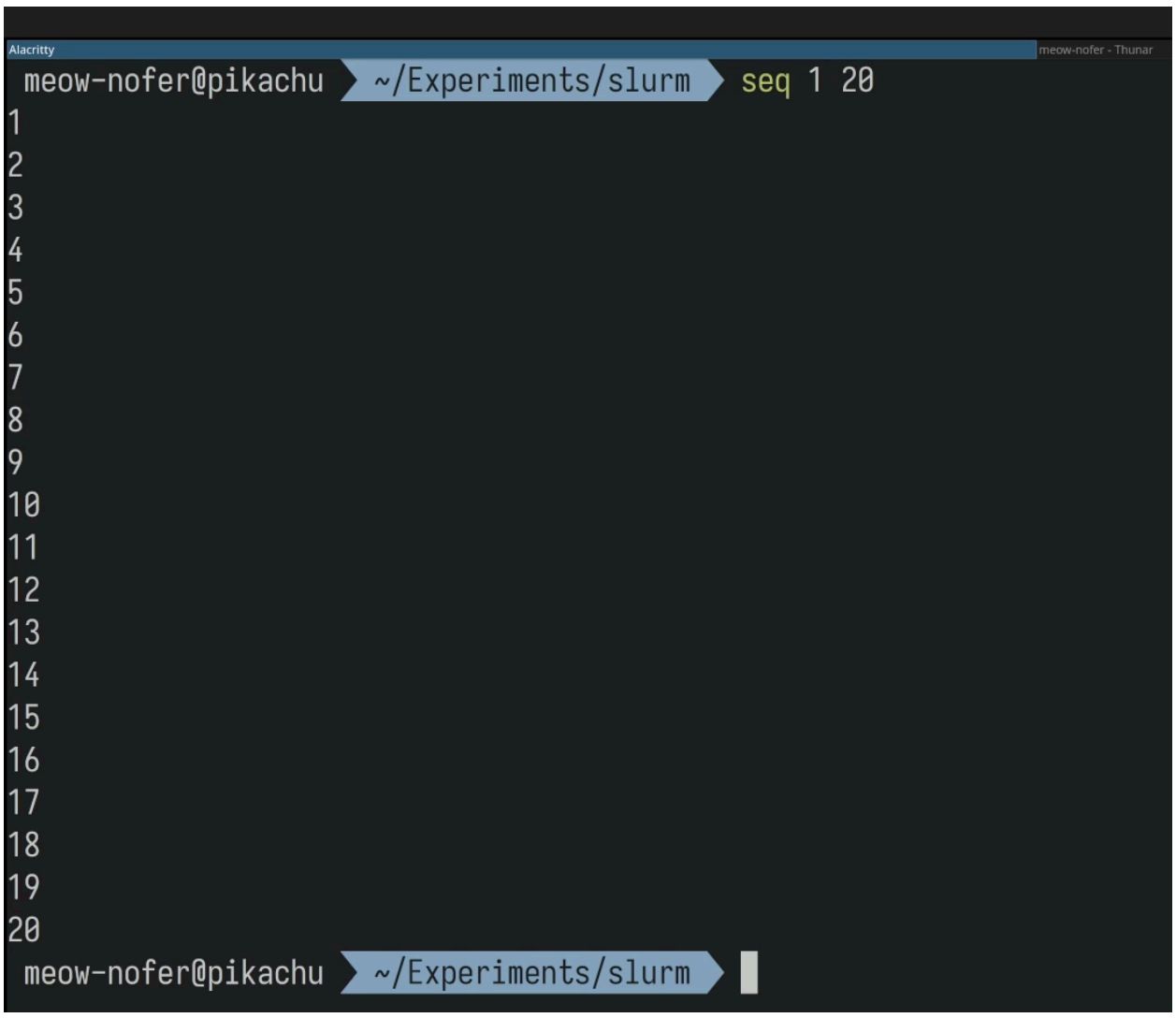


Все эти команды в той или иной степени вам знакомы.

Первые шаги

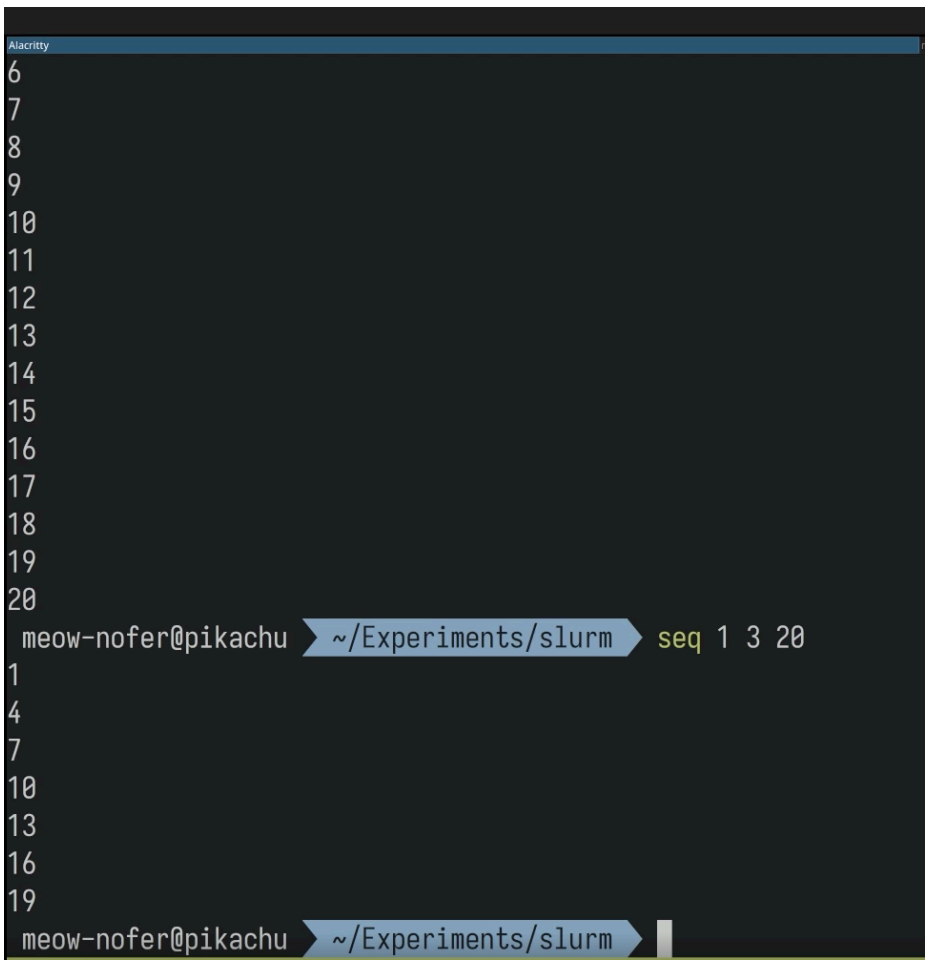
- **~$ seq [first [incr]] last #** последовательность чисел.

Выглядеть это может так:

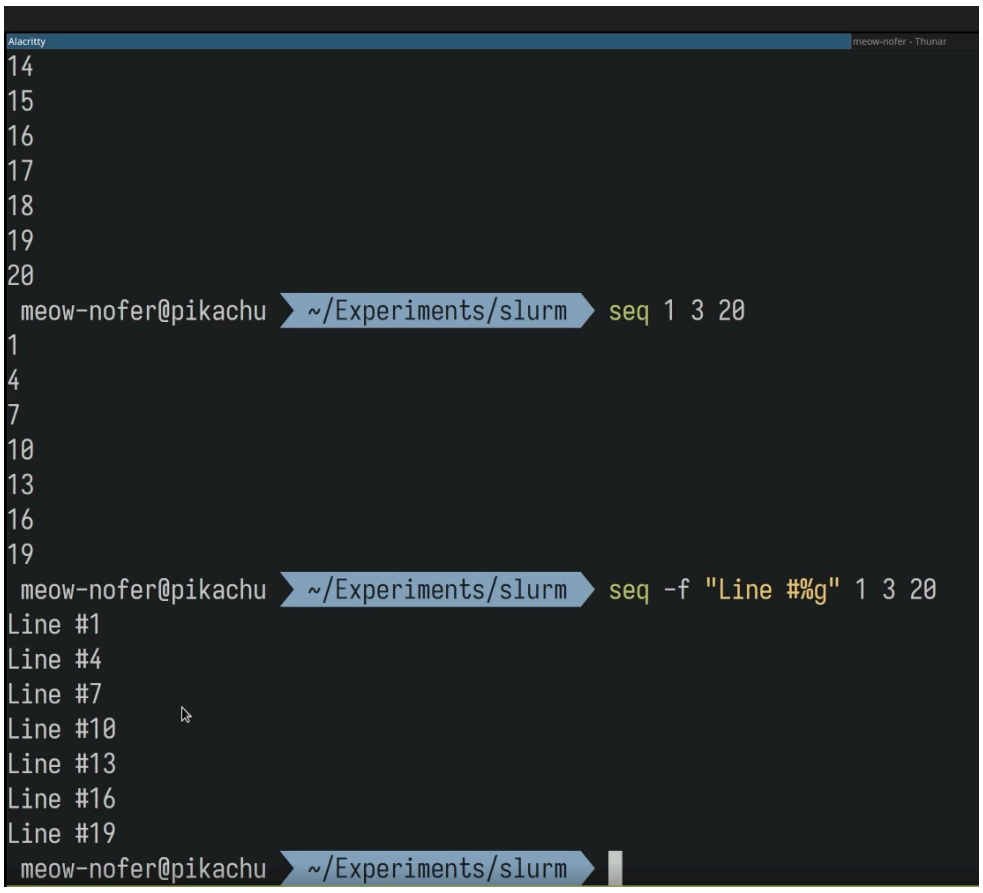


Мы пишем seq от «1» до «20» и получаем числа от «1» до «20».

Если мы хотим получить числа от «1» до «20» с шагом в «3», то в середине вставляем размер шага: 1, 3, 20:



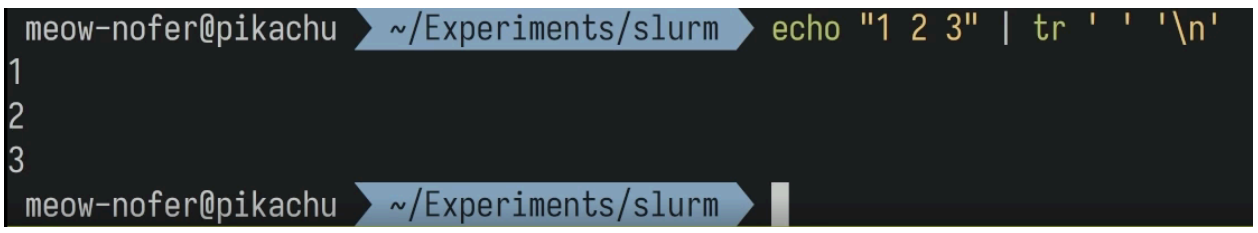
Также мы можем использовать дополнительное форматирование:



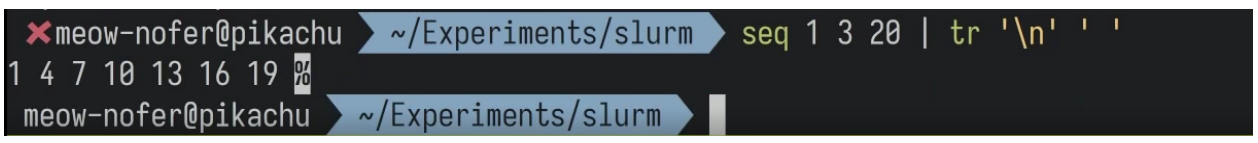
- ~$ tr ´first´´second´ # замена символов во входных строках.

Основная задача: заменять символы в исходной строке на символ, представленный в аргументе.

Например, у нас есть числа «1, 2, 3», и мы хотим получить из этого последовательность. Тогда мы tr заменяем на перенос строки:



Попробуем наоборот:

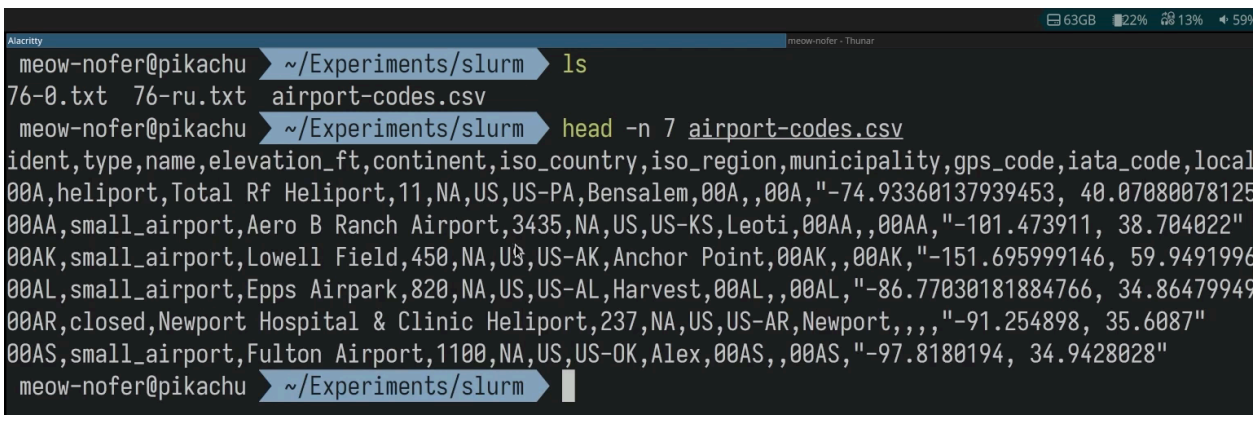


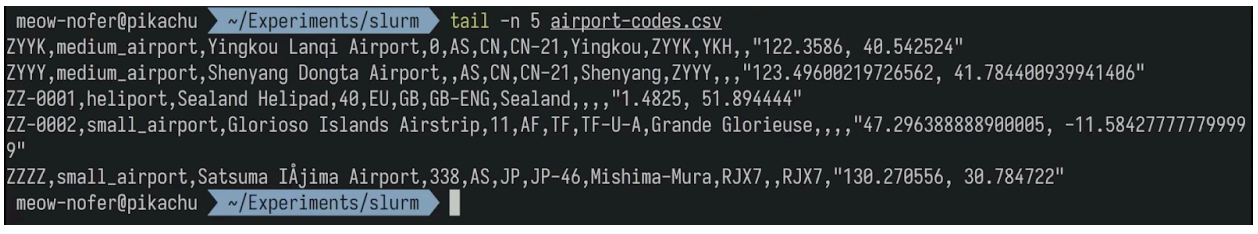
- **~$ zcat / gunzip -c #** распаковка файла с выводом на терминал.

Большинство, о чем мы рассказываем, применимо как к Linux, так и к macOS. Некоторые утилиты в macOS присутствуют, но имеют немного другой набор аргументов. **Gunzip** -c работает в двух вариантах.

- **~$ head -n 10 #** вывод первых нескольких строк; **~$ tail -n 10 #** вывод последних нескольких строк.

Рассмотрим на примере:





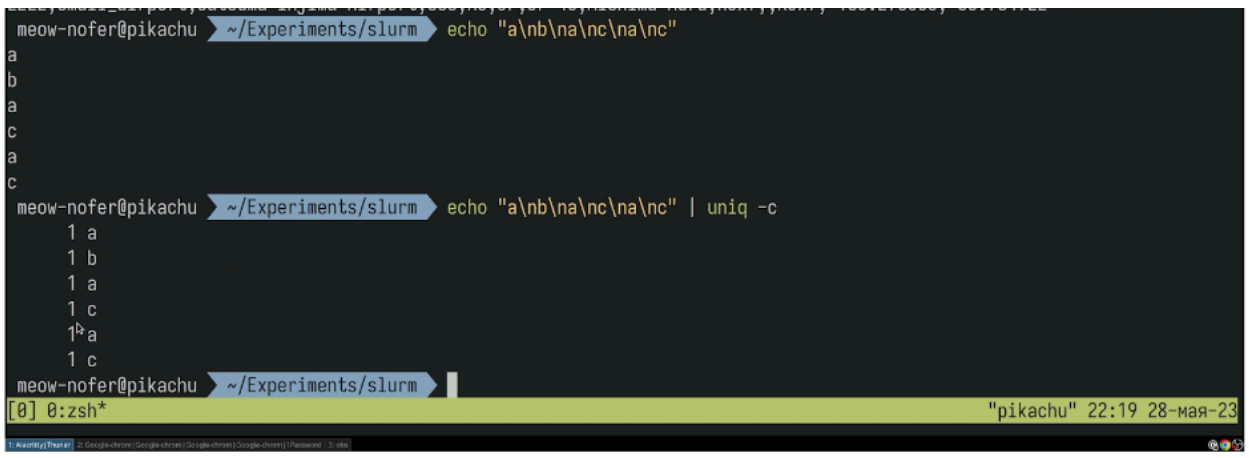
- **~$ tail -n+100 #** вывод последних строк, начиная со строки 100:

- **~$ zgrep** # grep по архивированным файлам;

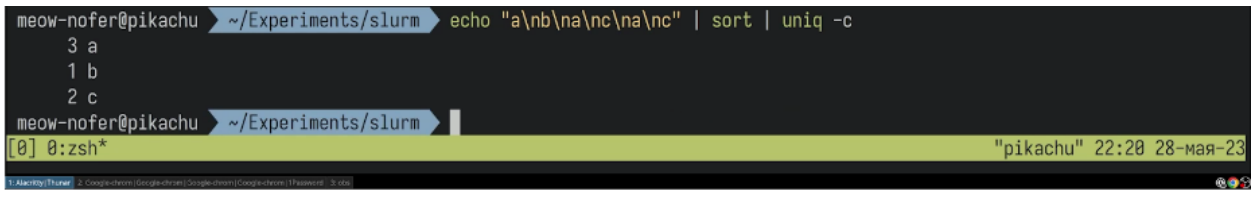
- **~$ uniq -c** # подсчет количества уникальных строк.

У данной команды есть «двойное дно».

Рассмотрим на примере:



Многие забывают, что uniq работает на отсортированных данных, то есть, он не хранит структуру в памяти, а просто схлопывает идущие подряд строчки. Поэтому, для корректности данные необходимо отсортировывать:



Sort умеет выгружать часть данных на диск. Он поможет с очень большими файлами.

Кодировка: enca и iconv

Если мы работаем в консоли с разными кодировками, то нам нужны утилиты:

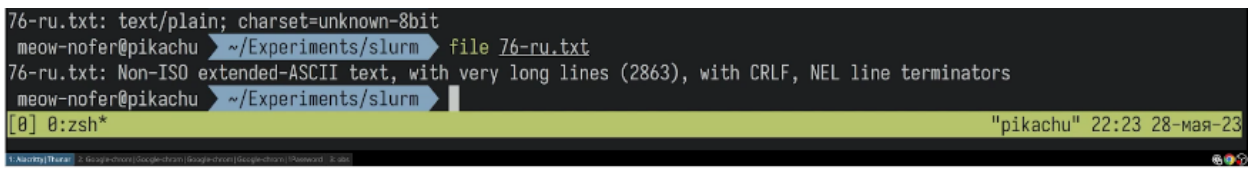
- Enca;
- Iconv.

Рассмотрим на примере:

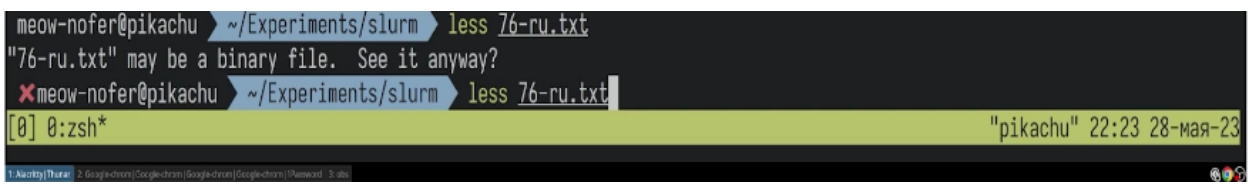
У нас есть два файла с текстом (в примере использован текст «Приключения Гекльберри Финна» М. Твена) на английском языке и на русском.

Команда «File» помогает узнать, с каким файлом мы работаем:

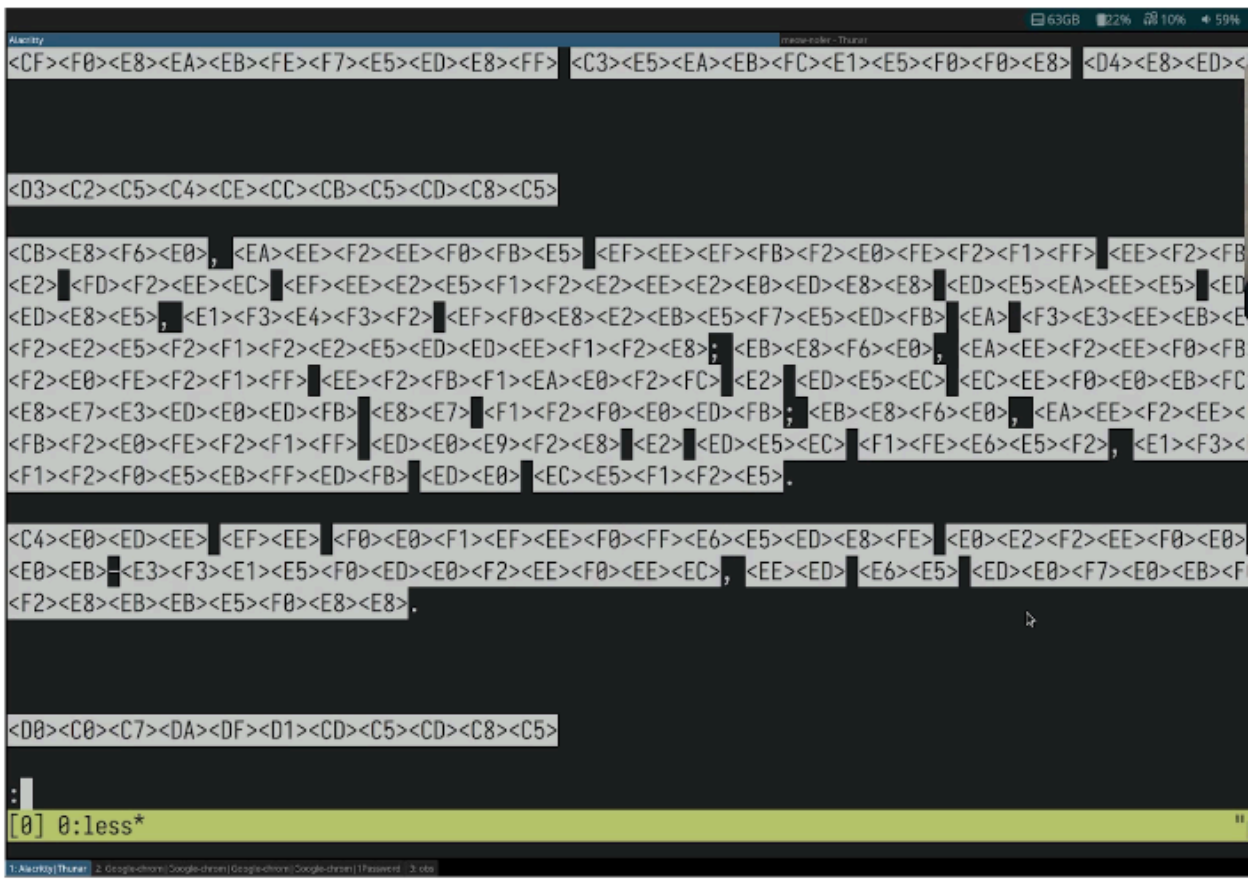
Если «натравить» команду на русскую версию, то получим следующее:



Пробуем командой «less» прочитать файл. Терминал сразу сообщит, что файл бинарный:



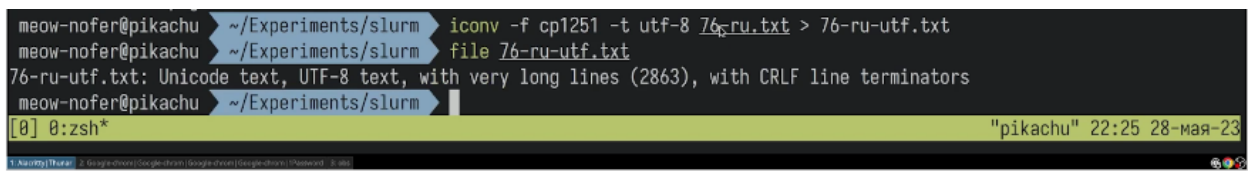
На самом деле файл в виндовой кодировке:



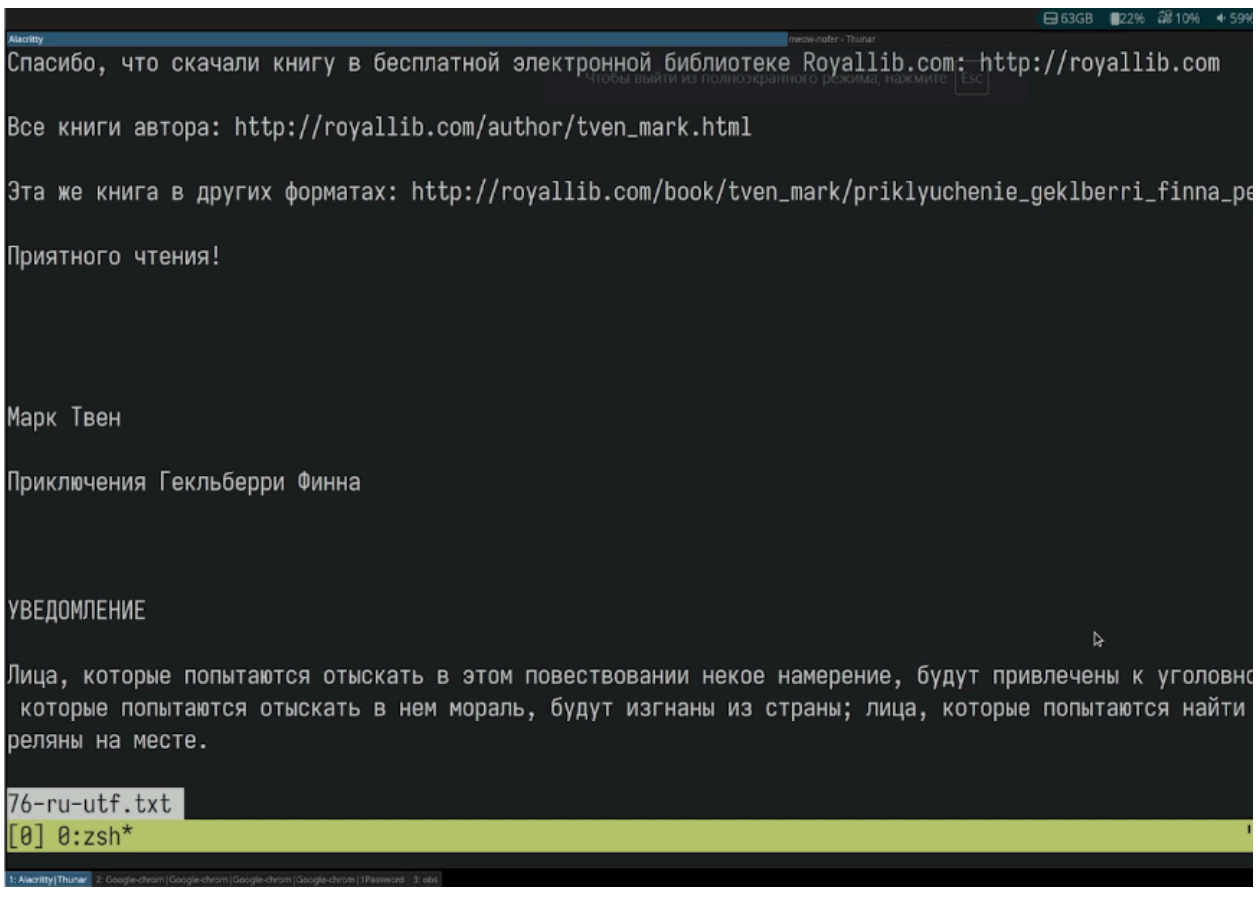
Что делаем: Enca помогает узнать кодировку файла. Пишем «**Enca -L ru 76-ru**».

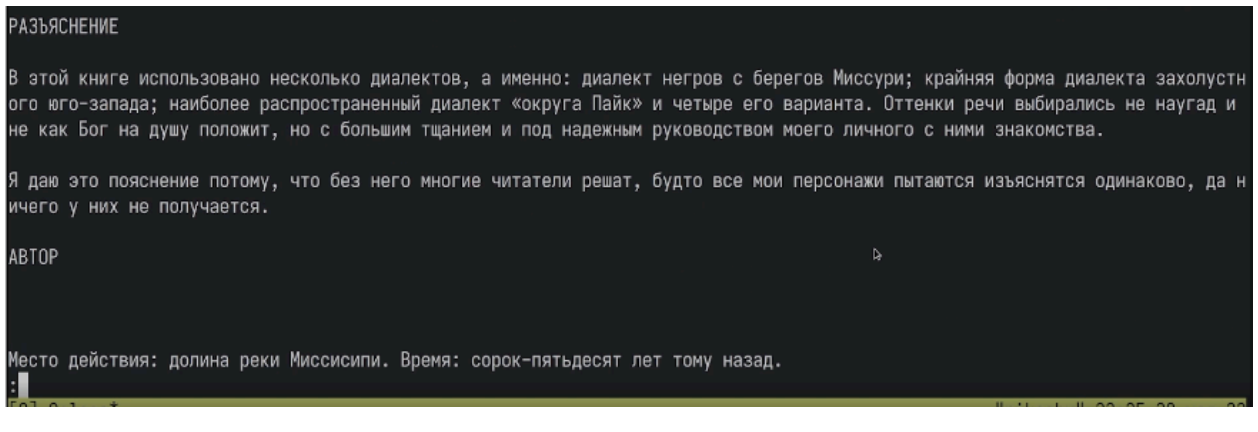
Иногда бывает так, что присланные файлы – файлы Windows. Для процессинга гораздо удобнее сконвертировать их в utf-8:

(с 13:07 по 13:32)



Если мы откроем этот файл в less, то все станет читаемым:





File всегда используется по умолчанию, а enca и iconv упрощают кодировку.

Трубоукладка

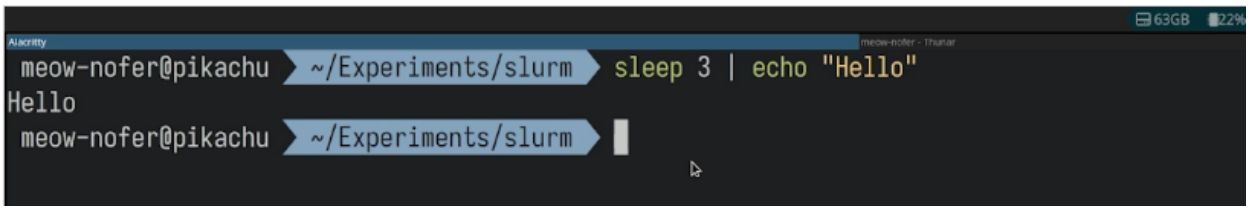
Одна из главных фич консоли Linux – **пайпинг** (когда вывод предыдущей команды может быть подан вводом для следующей команды).

Мы составляем пайплайн:

```
~$ cat file.csv | grep -v Title | tr ‘n’ ‘0’ | awk ‘{ print $5 }’
```

Мы запускаем **~$ cat file.csv**, исключаем оттуда все строчки, в которых есть «title», используем awk и выводим оттуда только пятую колонку.

Как это работает:



Мы можем сказать, например, «sleep 3», думая, что «echo “Hello”» вызовется позже. Это неверное представление. **Все команды, которые мы перечисляем в пайплайне запускаются одновременно**, после чего начинают обмениваться данными между собой.

Когда команда завершается, она делает это кодом выхода. Если код – 0, значит программа завершилась без нареканий, если же код не нулевой, то это возвещает об ошибке.

Если упадет какое-то промежуточное звено из команд (например, где-то в середине), нам все равно из всей конструкции

```
~$ cat file.csv | grep -v Title | tr ‘n’ ‘0’ | awk ‘{ print $5 }’
```

вернется код возврата последней команды в списке. Даная команда могла завершиться без ошибок. Это проблема!

Для решения этой проблемы существует «перехват кодов выхода». Его добавляют в начало скриптов: **~$ set -o pipefail**.

Далее, для объединения двух команд в выводе используют «**~$ {echo "1"; echo "2"} / other_command**.

Особенность, которую стоит учесть:

Если **grep** не нашел в файле ни одной строчки, которая не подпадает под переданный шаблон регулярного выражения, то он падает с ненулевым кодом возврата. Хак: (**grep "foo" // true) / less**.

Как вам урок? 😠 😐 🔥

Далее >