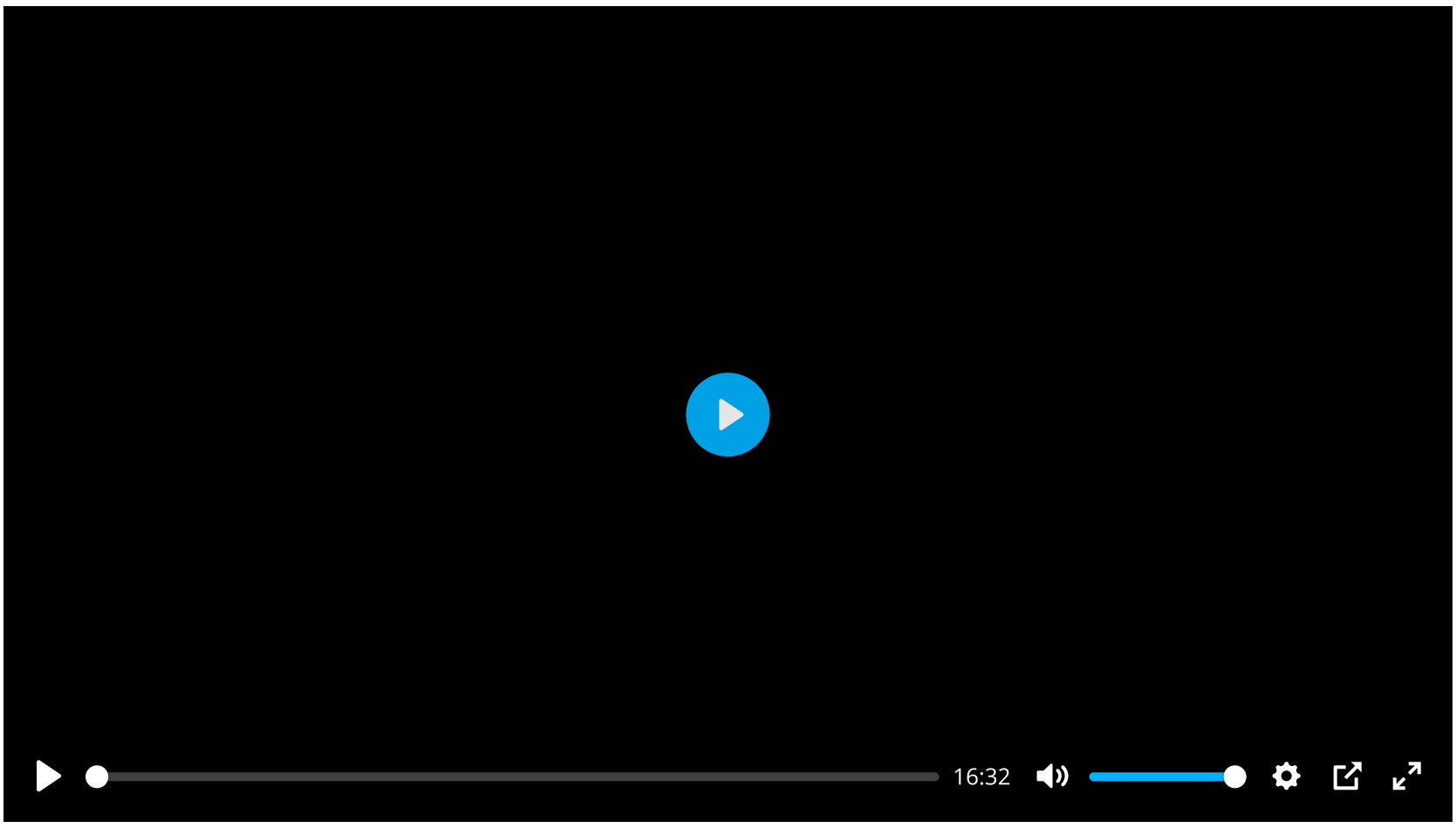
2.5 Linux терминал для задач DE



Текстовая расшифровка видео:

ПАРАЛЛЕЛИЗАЦИЯ ОБРАБОТКИ

План:

- XArgs как магический инструмент;
- Gnu Parallel – распределенка в консоли;
- Больше трюков;
- Список литературы.

XArgs как магический инструмент

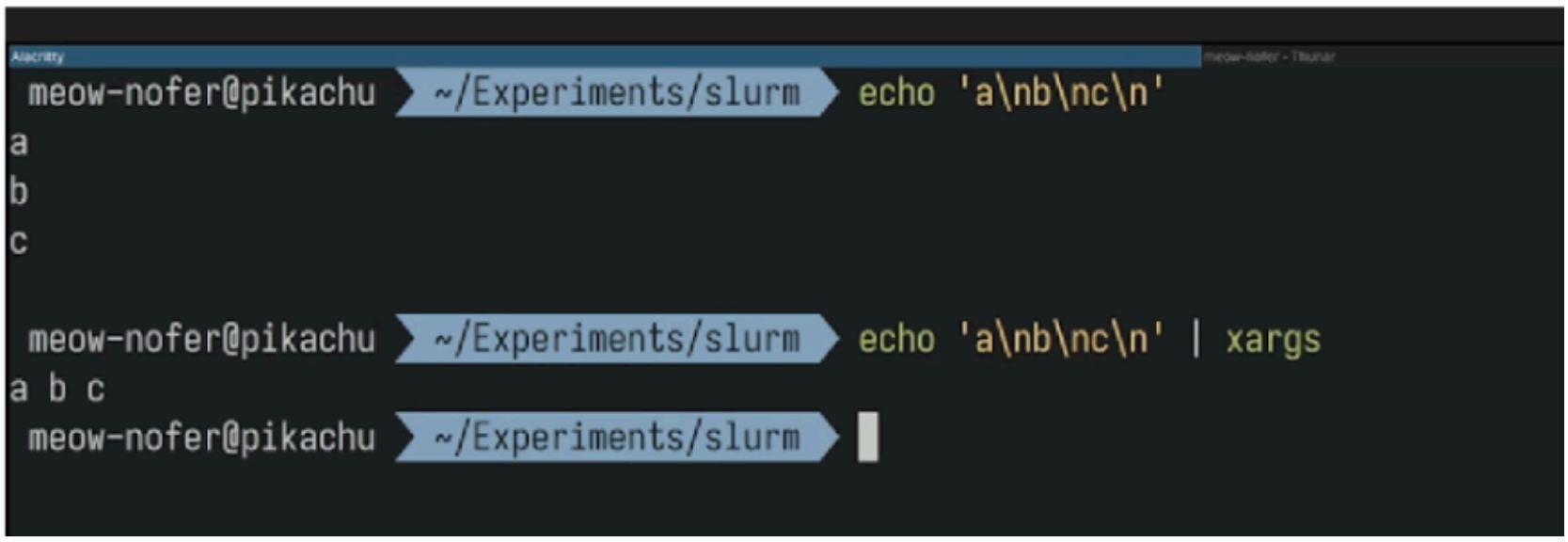
XArgs – это команда, которая принимает данные на вход и позволяет подставить эти данные в качестве аргумента для другой команды.

Рассмотрим пример:

В первом и втором примерах ищем файлы с расширением .mp3" и .sh". Вы можете заметить «-print0». Иногда бывает так, что файл именуется с переносом строки в названии. После того, как мы находим все mp3" и .sh" вызываем xargs и говорим «mv», вписываем все скрипты, которые нашли, а после пишем, например, .sh в каталог /back.scripts.

Также XArgs имеет возможность убирать переносы строк.

Рассмотрим пример:



Например, если у вас есть файл с переносами строк, то можно «скормить» его XArgs без аргументов, и он уберет эти переносы. По умолчанию команда называется «echo».

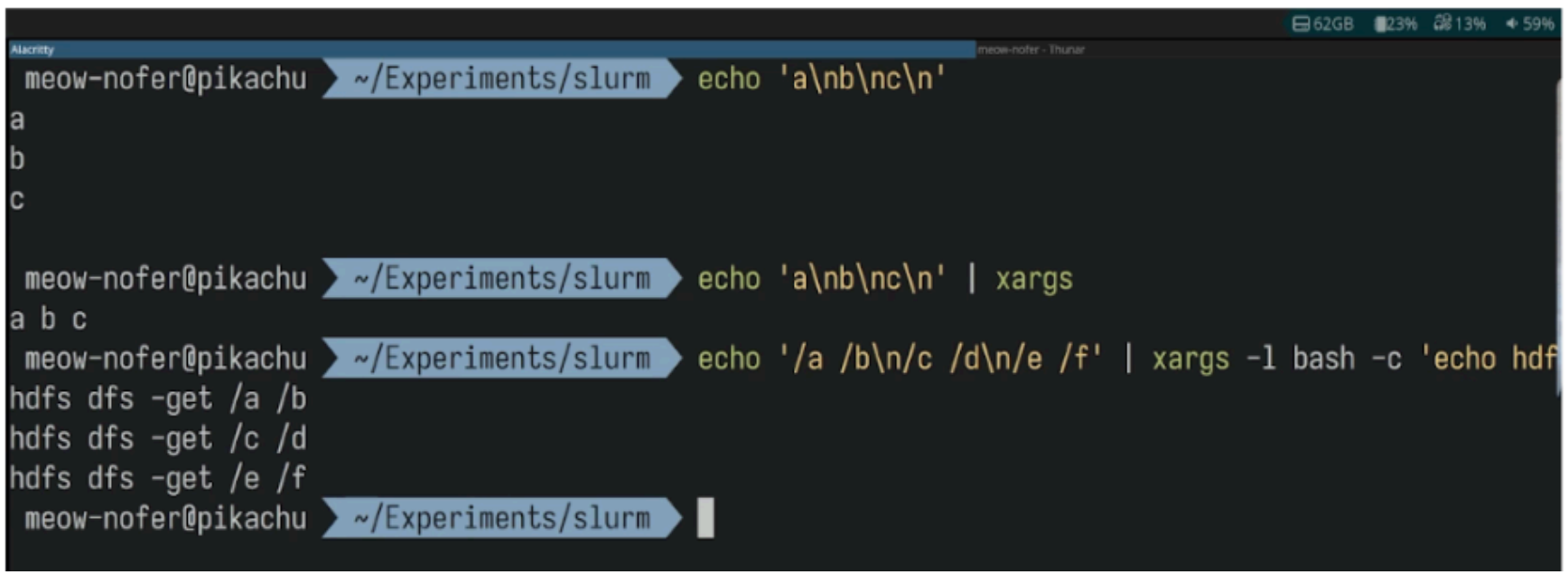
У XArgs есть возможность запуска **более одного экземпляра внутри себя.**

Вопрос: что делать, если мы хотим его распараллелить?

Например, нам нужно выкачать файлы с HDFS и переложить их по локальному пути. Подобную задачу можно решить двумя XArgs:

```
~$ cat file.txt | xargs -I bash -c 'echo hdfs dfs -get $0 $1' | xargs -I {} -d '\n' -n1 -P8 -t bash -c "eval {}"
```

Первый XArgs довольно простой:



В исходном файле через пробел написано несколько путей. Мы **/a** из HDFS перемещаем в **/b** локально; **/c** в **/d**; **/e** в **/f**. Далее, с помощью XArgs говорим «сгенерируй bash-команду HDFS get откуда-куда». Так, у нас запустится набор команд. Далее, запускаем команды. Также мы можем копировать файлы в параллель. Чтобы это не занимало много времени, можно вызвать XArgs, задав больше параметров. По умолчанию мы запускаем восемь процессов, **каждому из них мы «скармливаем» по одной строке.** Далее, вызываем «bash -c "eval»:

```
~$ cat file.txt | xargs -I bash -c 'echo hdfs dfs -get $0 $1' | xargs -I {} -d '\n' -n1 -P8 -t bash -c "eval {}"
```

Gnu Parallel – распределенка в консоли

Gnu Parallel – маленький скрипт, написанный на Perl. Отличается высокой скоростью.

Если нам необходимо процессить большой файл в параллель, то стоит делать это в параллельной консоли. Это на порядок быстрее, чем в Python, Pandas, потоками или процессами.

Рассмотрим простые примеры:

```
~$ ls *.wav | parallel lame {} -o {}.mp3

~$ python makelist.py | parallel -j+2 'wget "{}" -O - | python parse.py'

~$ cat file.txt | awk '{ print "{\"index\": {} }","\n" $0 }' | parallel --pipe -N500 curl -s -XPOST
localhost:9200/items/entry/_bulk --data-binary @- > /dev/null

man parallel_tutorial
```

Здесь мы можем кодировать сразу пачку mp3-файлов.

Стоит обратить внимание на **-j**. Он указывает на то, сколько параллельных процессов мы хотим запустить. По умолчанию запускается то количество процессов, сколько у нас ядер в процессоре. Здесь же можно приплюсовать еще два процесса. Так, например, мы в Parallel скачиваем пачку файлов.

Результат скрипта Python – сгенерировать пачку URL. Мы в Parallel вызываем wget, и у нас одновременно качается большое количество файлов с wget'ом. Мы его переадресуем для каждого файла в специальный парсер на Python. После чего сохраняем или перенаправляем.

Больше трюков!

Существует большое количество трюков:

```
~$ cat file.csv | parallel --colsep "\t" echo {2} {1} {3}

~$ cat file.csv | awk 'la[$1]++'  # sort -u не нужно

~$ cat file.csv | pv --line-mode -b > /dev/null  # ОЧЕНЬ КРУТО

~$ cat file.csv | peco | some_other_process  # выбор строк вручную

~$cat 76-0.txt | tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort | uniq -c | sort -nr -k1,1 | head
-n 50 | awk '{ print $2 "\t" $1 }' | gnuplot -p -e "set term png; set xtic rotate; plot '-' using
(column(0)):2:xtic(1) smooth freq with boxes" > test.png
```

Например, с помощью Parallel мы можем сортировать колонки.

Также мы говорили про sort и uniq, где **sort** сортирует блочно, а **uniq** – схлопывает последовательно. Обратите внимание на реализацию uniq на awk, которая представляет собой «черную магию»: a – автоматически воспринимается как словарь в данных. Поскольку это динамическая типизация, мы добавляем туда строчки, которые прилетают из стандартного ввода (если они есть, добавляем; если их нет, то значение по этой строчке в словаре увеличиваем на единицу).

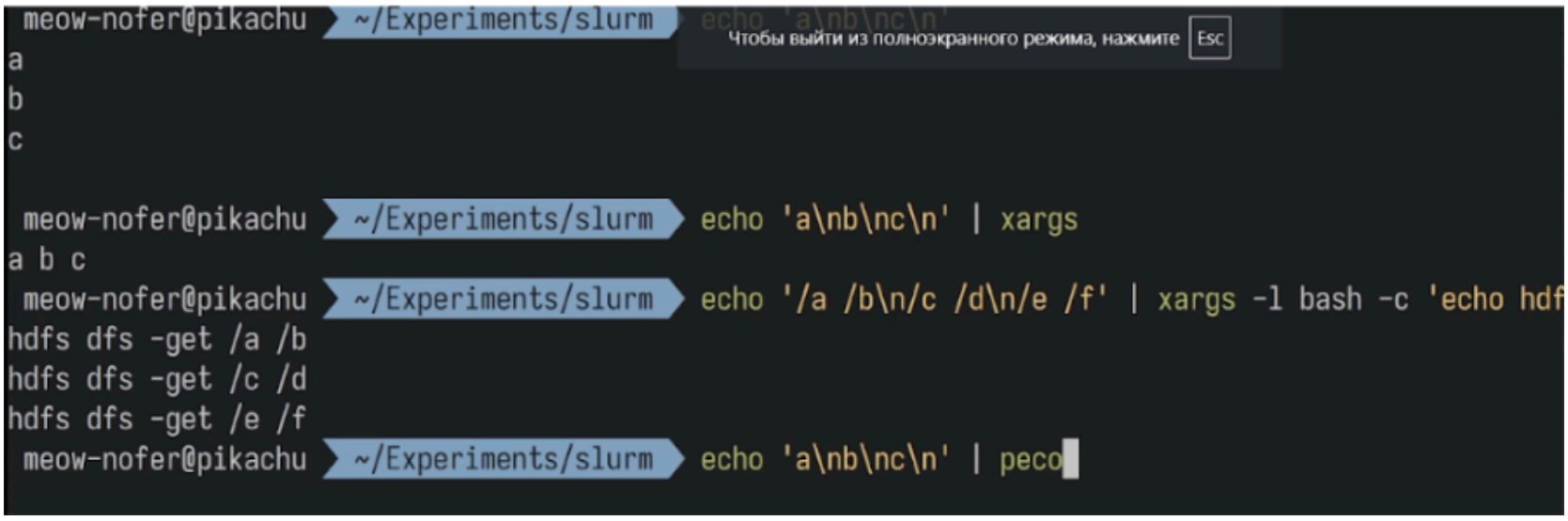
Вопрос: как отладить длинные пайплайны? Как наблюдать за прогрессом?

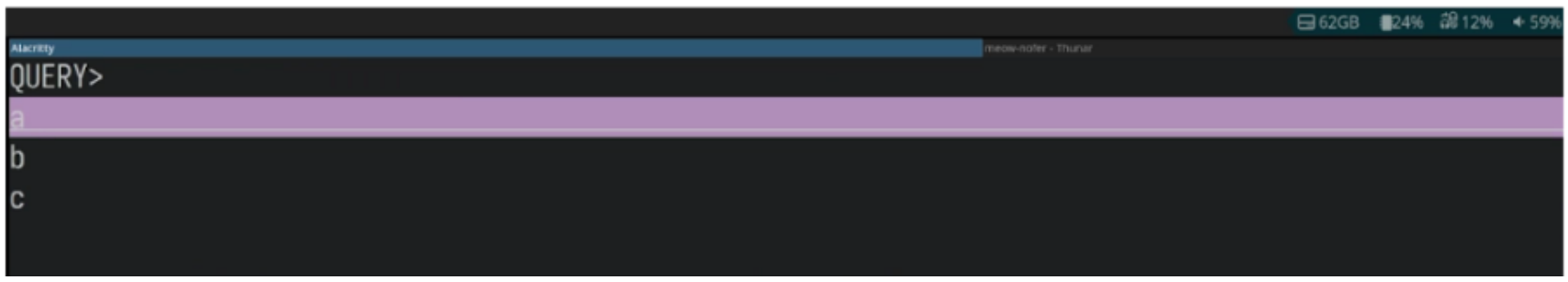
Ответ: для этого существует утилита, которая называется «**pv**».

«**Pv**» – это утилита, которая показывает обновляющееся количество того, сколько строчек/байтов через нее прошло.

Стоит сказать и о «peco», когда нужно в процессе пайплайна получить пользовательский ввод, чтобы пользователь самостоятельно смог выбрать, какие данные нужно отправить дальше.

Рассмотрим на примере:



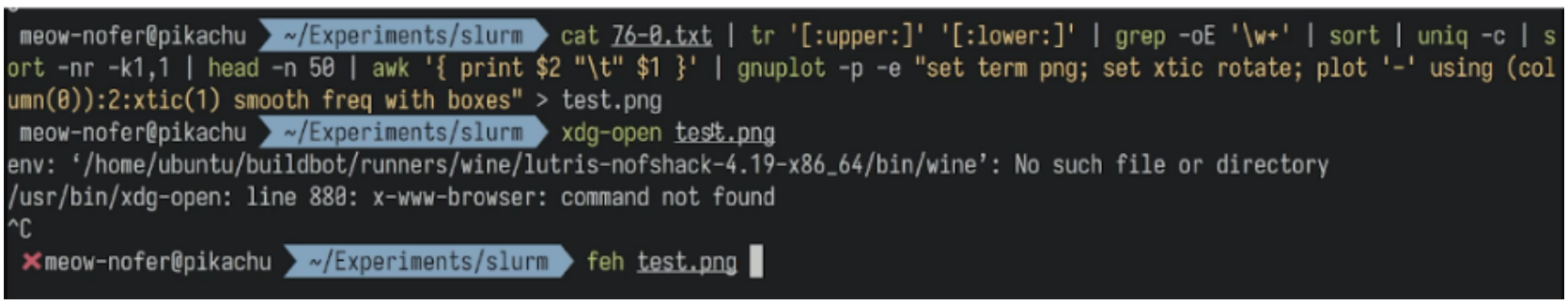


Мы передаем строчки (peco позволяет выбирать несколько) и выбираем, какие нужно отправить. Это примитивно, но удобно в использовании.

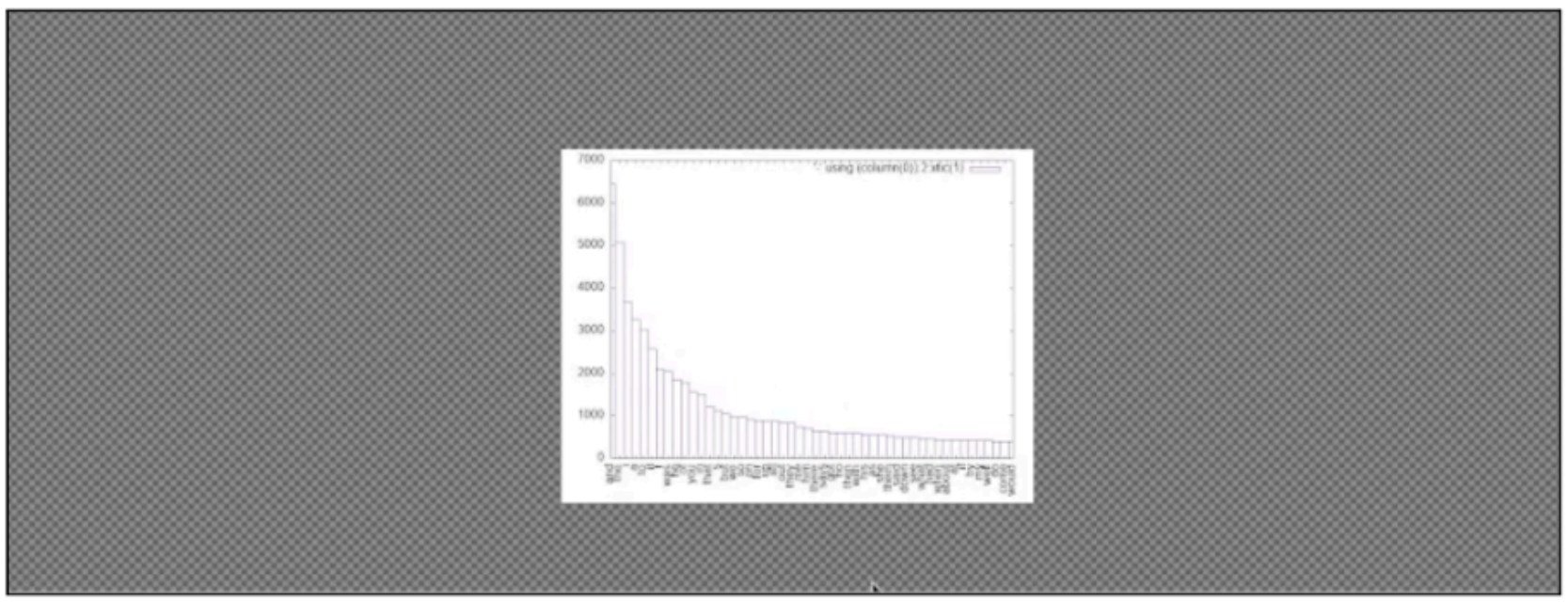
Вопрос: чем можно заменить графики?

Ответ: есть консольная утилита, которая старше, чем все новомодные plot, seaborn и т.д. (это библиотеки в Data Scientist для рисования графиков). Мы можем их использовать для генерации файлов с графикой, даже если на сервере нет иксов.

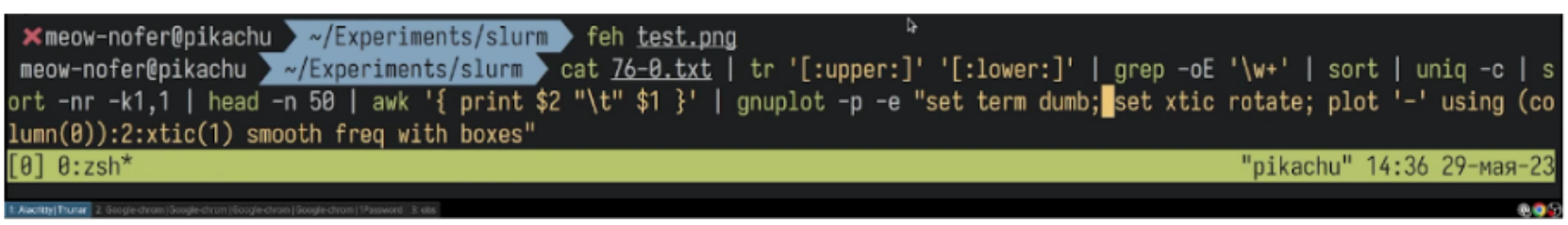
Выглядит это следующим образом:



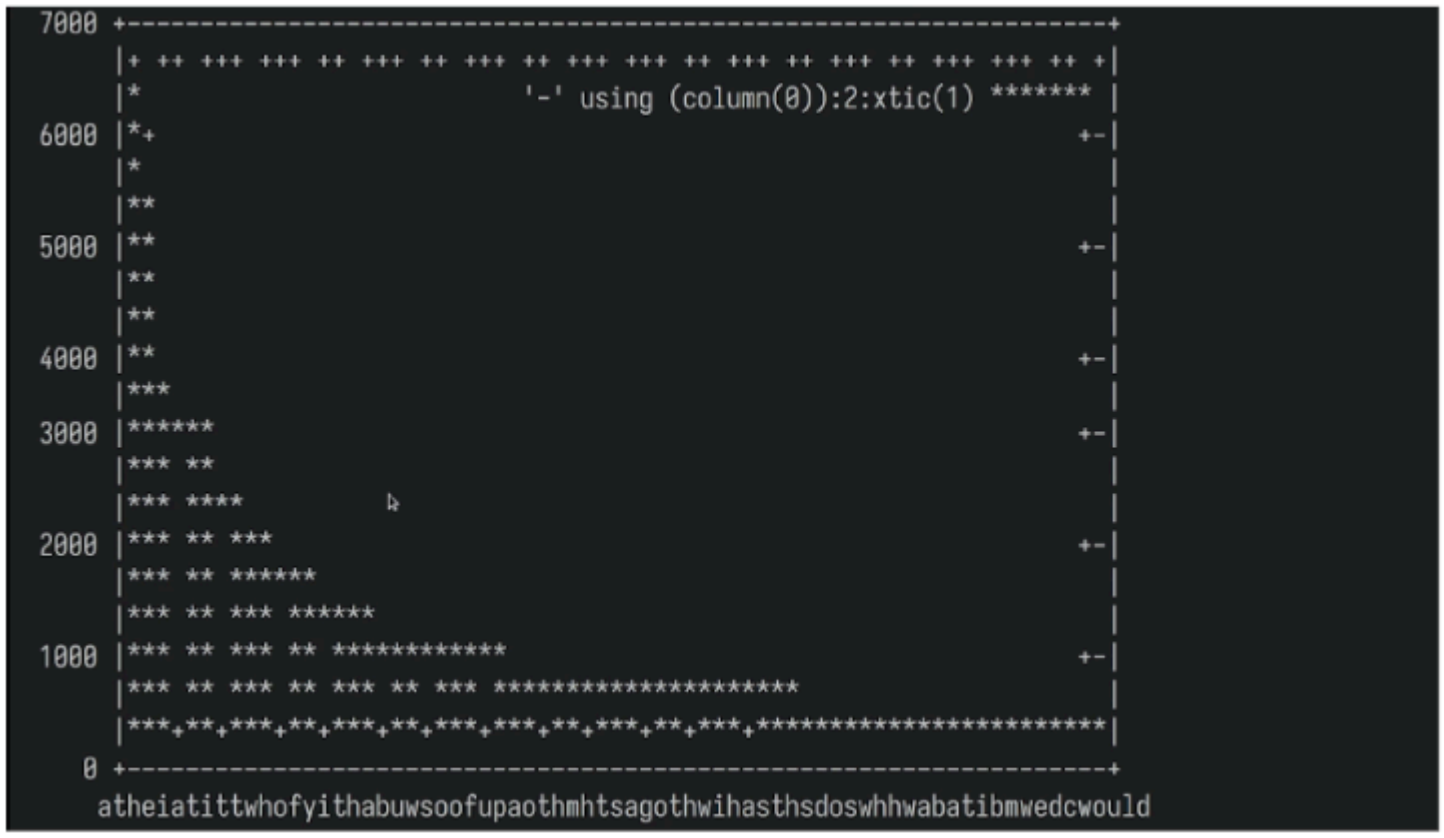
Если мы выполним этот код и посмотрим, что у нас получилось в файле test.png, то получится график по словам:



Можно сделать еще веселее. Мы можем в **set term** указать «dumb»:



Вместо картинки получится абсолютно нечитаемая нижняя строка, но, по сути, это полноценный график, который может что-либо иллюстрировать:



Список литературы

Предлагаем ознакомиться с ссылками на подборки разных утилит для работы с данными и не только:

- https://github.com/jlevy/the-art-of-command-line
- https://github.com/agarrharr/awesome-cli-apps
- https://github.com/learn-anything/command-line-tools
- https://xakep.ru/2016/05/17/console-magic/
- https://xakep.ru/2016/06/07/console-magic-2/

Как вам урок?

Далее