

Текстовая расшифровка видео:

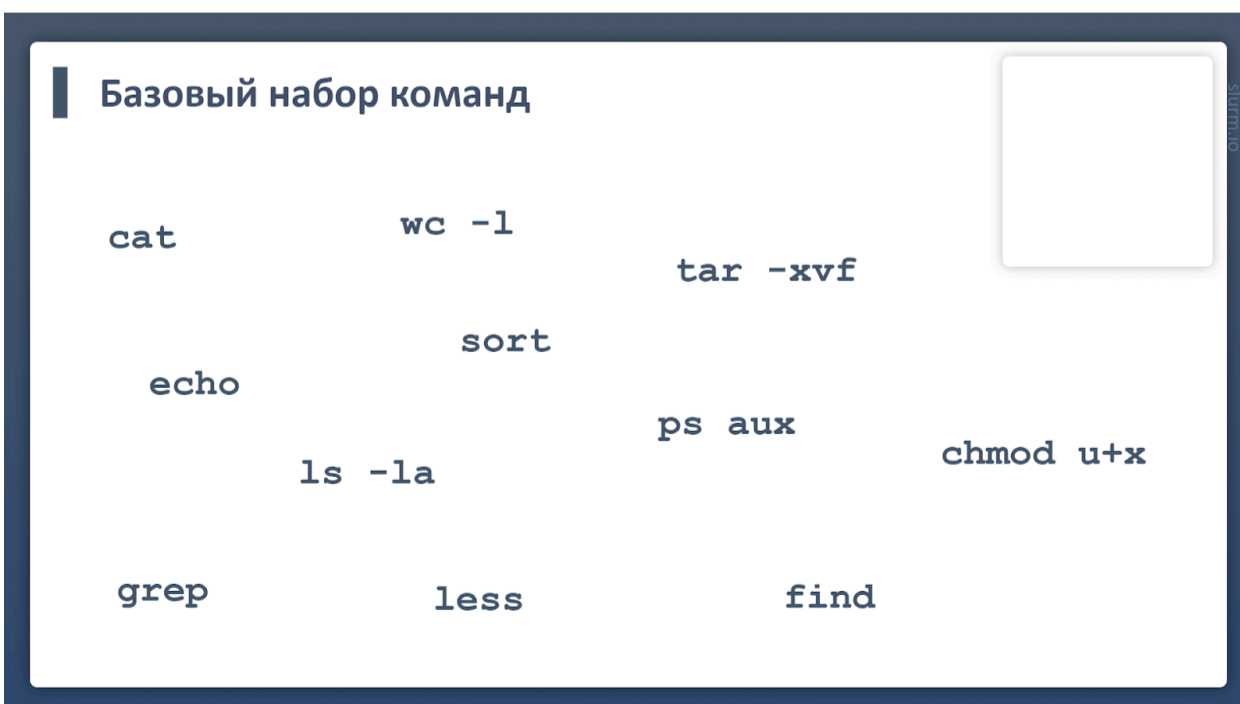
ИТАК, У МЕНЯ ЕСТЬ ТЕКСТОВЫЕ ДАННЫЕ...

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:

```
meow-nofer@pikachu ~/Experiments/slurm seq 1 20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
meow-nofer@pikachu ~/Experiments/slurm
```

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:

```
meow-nofer@pikachu ~/Experiments/slurm seq 1 3 20
1
4
7
10
13
16
19
meow-nofer@pikachu ~/Experiments/slurm
```

Также мы можем использовать дополнительное форматирование:

```
meow-nofer@pikachu ~/Experiments/slurm
14
15
16
17
18
19
20
meow-nofer@pikachu ~/Experiments/slurm seq 1 3 20
1
4
7
10
13
16
19
meow-nofer@pikachu ~/Experiments/slurm seq -f "Line #%" 1 3 20
Line #1
Line #4
Line #7
Line #10
Line #13
Line #16
Line #19
meow-nofer@pikachu ~/Experiments/slurm
```

- `~$ 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:

```
meow-nofer@pikachu ~/Experiments/slurm echo "1 2 3" | tr ' ' '\n'
1
2
3
meow-nofer@pikachu ~/Experiments/slurm
```

Попробуем наоборот:

```
meow-nofer@pikachu ~/Experiments/slurm seq 1 3 20 | tr '\n' ' '
1 4 7 10 13 16 19
meow-nofer@pikachu ~/Experiments/slurm
```

- `~$ 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** -с работает в двух вариантах.

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

Рассмотрим на примере:

```
meow-nofer@pikachu ~/Experiments/slurm ls
76-0.txt 76-ru.txt airport-codes.csv
meow-nofer@pikachu ~/Experiments/slurm head -n 7 airport-codes.csv
ident,type,name,elevation_ft,continent,iso_country,iso_region,municipality,gps_code,iata_code,local
00A,heliport,Total Rf Heliport,11,NA,US,US-PA,Bensalem,00A,,00A,"-74.93360137939453, 40.07080078125
00AA,small_airport,Aero B Ranch Airport,3435,NA,US,US-KS,Leoti,00AA,,00AA,"-101.473911, 38.704022"
00AK,small_airport,Lowell Field,450,NA,US,US-AK,Anchor Point,00AK,,00AK,"-151.695999146, 59.9491996
00AL,small_airport,Epps Airpark,820,NA,US,US-AL,Harvest,00AL,,00AL,"-86.77030181884766, 34.86479949
00AR,closed,Newport Hospital & Clinic Heliport,237,NA,US,US-AR,Newport,,,"-91.254898, 35.6087"
00AS,small_airport,Fulton Airport,1100,NA,US,US-OK,Alex,00AS,,00AS,"-97.8180194, 34.9428028"
meow-nofer@pikachu ~/Experiments/slurm

meow-nofer@pikachu ~/Experiments/slurm tail -n 5 airport-codes.csv
ZYYK,medium_airport,Yingkou Lanqi Airport,0,AS,CN,CN-21,Yingkou,ZYYK,YKH,"122.3586, 40.542524"
ZYYY,medium_airport,Shenyang Dongta Airport,,AS,CN,CN-21,Shenyang,ZYYY,,,"123.49600219726562, 41.784400939941406"
ZZ-0001,heliport,Sealand Helipad,40,EU,GB,GB-ENG,Sealand,,,"1.4825, 51.894444"
ZZ-0002,small_airport,Glorioso Islands Airstrip,11,AF,TF,TF-U-A,Grande Glorieuse,,,"47.296388888900005, -11.5842777779999
9"
ZZZZ,small_airport,Satsuma Iijima Airport,338,AS,JP,JP-46,Mishima-Mura,RJX7,,RJX7,"130.270556, 30.784722"
meow-nofer@pikachu ~/Experiments/slurm
```

- `~$ 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:

```
meow-nofer@pikachu ~/Experiments/slurm echo "a\nb\na\nc\na\nc"
a
b
a
c
a
c
meow-nofer@pikachu ~/Experiments/slurm echo "a\nb\na\nc\na\nc" | uniq -c
 1 a
 1 b
 1 a
 1 c
 1 a
 1 c
meow-nofer@pikachu ~/Experiments/slurm
```

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 отсортировать:

```
meow-nofer@pikachu ~/Experiments/slurm echo "a\nb\na\nc\na\nc" | sort | uniq -c
 3 a
 1 b
 2 c
meow-nofer@pikachu ~/Experiments/slurm
```

`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:

```
meow-nofer@pikachu ~/Experiments/slurm ls
76-0.txt 76-ru.txt airport-codes.csv
meow-nofer@pikachu ~/Experiments/slurm file 76-0.txt
76-0.txt: Unicode text, UTF-8 (with BOM) text, with CRLF line terminators
meow-nofer@pikachu ~/Experiments/slurm file -i 76-0.txt
76-0.txt: text/plain; charset=utf-8
meow-nofer@pikachu ~/Experiments/slurm
```

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

```
76-ru.txt: text/plain; charset=unknown-8bit
meow-nofer@pikachu ~/Experiments/slurm file 76-ru.txt
76-ru.txt: Non-ISO extended-ASCII text, with very long lines (2863), with CRLF, NEL line terminators
meow-nofer@pikachu ~/Experiments/slurm
```

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

```
meow-nofer@pikachu ~/Experiments/slurm less 76-ru.txt
"76-ru.txt" may be a binary file. See it anyway?
meow-nofer@pikachu ~/Experiments/slurm less 76-ru.txt
```

На самом деле файл в виндовой кодировке:

```
<CF><F0><E8><EA><EB><FE><F7><E5><ED><E8><FF> <C3><E5><EA><EB><FC><E1><E5><F0><F0><E8> <D4><E8><ED><
<D3><C2><C5><C4><CE><CC><CB><C5><CD><C8><C5>
<CB><E8><F6><E0>, <EA><EE><F2><EE><F0><FB><E5> <EF><EE><EF><FB><F2><E0><FE><F2><F1><FF> <EE><F2><FB
<E2> <FD><F2><EE><EC> <EF><EE><E2><E5><F1><F2><E2><EE><E2><E0><ED><E8><E8> <ED><E5><EA><EE><E5> <ED
<ED><E8><E5> <E1><F3><E4><F3><F2> <EF><F0><E8><E2><EB><E5><F7><E5><ED><FB> <EA><F3><E3><EE><EB><E
<F2><E2><E5><F2><F1><F2><E2><E5><ED><ED><EE><F1><F2><E8> <EB><E8><F6><E0>, <EA><EE><F2><EE><F0><FB
<F2><E0><FE><F2><F1><FF> <EE><F2><FB><F1><EA><E0><F2><FC> <E2> <ED><E5><EC> <EC><EE><F0><E0><EB><FC
<E8><E7><E3><ED><E0><ED><FB> <E8><E7> <F1><F2><F0><E0><ED><FB> <EB><E8><F6><E0>, <EA><EE><F2><EE><
<FB><F2><E0><FE><F2><F1><FF> <ED><E0><E9><F2><E8> <E2> <ED><E5><EC> <F1><FE><E6><E5><F2>, <E1><F3><
<F1><F2><F0><E5><EB><FF><ED><FB> <ED><E0> <EC><E5><F1><F2><E5>.
<C4><E0><ED><EE> <EF><EE> <F0><E0><F1><EF><EE><F0><FF><E6><E5><ED><E8><FE> <E0><E2><F2><EE><F0><E0>
<E0><EB> <E3><F3><E1><E5><F0><ED><E0><F2><EE><F0><EE><EC>, <EE><ED> <E6><E5> <ED><E0><F7><E0><EB><F
<F2><E8><EB><EB><E5><F0><E8><E8>.
<D0><C0><C7><DA><DF><D1><CD><C5><CD><C8><C5>
:
[0] 0:less*
```

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)

```
meow-nofer@pikachu ~/Experiments/slurm iconv -f cp1251 -t utf-8 76-ru.txt > 76-ru-utf.txt
meow-nofer@pikachu ~/Experiments/slurm file 76-ru-utf.txt
76-ru-utf.txt: Unicode text, UTF-8 text, with very long lines (2863), with CRLF line terminators
meow-nofer@pikachu ~/Experiments/slurm
[0] 0:zsh* "pikachu" 22:25 28-мая-23

meow-nofer@pikachu ~/Experiments/slurm enca -L ru 76-ru-utf.txt
Universal transformation format 8 bits; UTF-8
CRLF line terminators
meow-nofer@pikachu ~/Experiments/slurm
[0] 0:zsh* "pikachu" 22:25 28-мая-23
```

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

```
Спасибо, что скачали книгу в бесплатной электронной библиотеке Royallib.com: http://royallib.com
Все книги автора: http://royallib.com/author/tven_mark.html
Эта же книга в других форматах: http://royallib.com/book/tven_mark/priklyuchenie_geklberri_finna_pe
Приятного чтения!

Марк Твен
Приключения Гекльберри Финна

УВЕДОМЛЕНИЕ
Лица, которые попытаются отыскать в этом повествовании некое намерение, будут привлечены к уголовно
которые попытаются отыскать в нем мораль, будут изгнаны из страны; лица, которые попытаются найти
реляны на месте.

76-ru-utf.txt
[0] 0:zsh*
```

```
РАЗЪЯСНЕНИЕ
В этой книге использовано несколько диалектов, а именно: диалект негров с берегов Миссури; крайняя форма диалекта захолусти
ого юго-запада; наиболее распространенный диалект «округа Пайк» и четыре его варианта. Оттенки речи выбирались не наугад и
не как Бог на душу положит, но с большим тщанием и под надежным руководством моего личного с ними знакомства.

Я даю это пояснение потому, что без него многие читатели решат, будто все мои персонажи пытаются изъясняться одинаково, да н
ичего у них не получается.

АВТОР
Место действия: долина реки Миссисипи. Время: сорок-пятьдесят лет тому назад.
```

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

Трубоукладка

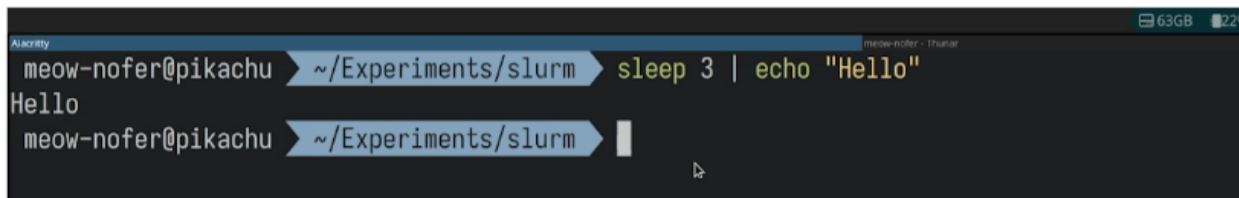
Одна из главных фиш консоли Linux – **пайпинг** (когда вывод предыдущей команды может быть подан вводом для следующей команды).

Мы составляем пайплайн:

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

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

Как это работает:



```
meow-nofer@pikachu ~/Experiments/slurm$ sleep 3 | echo "Hello"
Hello
meow-nofer@pikachu ~/Experiments/slurm$
```

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. Данная 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 строки, которая не подпадает под переданный шаблон регулярного выражения, то он падает с ненулевым кодом возврата. Хак: `(grep "foo" // true) / less`.

Как вам урок?



Далее >