

UniSat 01 --Введение Bash (Command Line Interface)on Unix (Linux)

Написана @ Sat 04 Jan 2020 10:09:30 PM +06 с помощью ❤️ Azat @ AzatAI

Перевод с английского @ Fri 28 Feb 2020 10:09:30 PM +06 с помощью ❤️ Kenges Yeldana Bazarkhanov Darkhan Kalniyazov Nurzhan Barat Bekzat @ AzatAI

Проверил перевод Boranbayev Zhandos @ AzatAI

UniSat 01 --Введение Bash (Command Line Interface)on Unix (Linux)

Общие правила

Быстрый старт

Базовая навигация

`pwd` -- это сообщает вам, о текущем рабочем каталоге.

`ls` -- затем мы захотим узнать, что там есть

Путь

`cd` Смена каталогов - перейти в другой каталог.

Задания

Файлы

Все - это файл.

Linux - это система без расширения

`file [path]` - получить информацию о том, какой тип файла файл или каталог.

Linux чувствителен к регистру символов

Spaces in names

Скрытые файлы и каталоги.

`ls -a` Перечислит содержимое каталога, включая скрытые файлы.

Краткое содержание

Задание

Справочная страница

Вступление

Каковы они?

Поисковик

`Long` и `Short` варианты

Задание

Манипуляция файлами

`mkdir` Создание каталога

`mkdir [-p]` при необходимости создайте родительские каталоги

`mkdir [-v]` печать сообщения для каждого созданного каталога.

`rmdir` Удаление каталога.

`touch` Создание пустого файла

`cp` Копирование файла или каталога

`mv` Перемещение файла или каталога

`mv` Переименование файлов и каталогов

`rm` Удаление файла (и непустых каталогов)

`rm -r` Удаление непустых каталогов

Краткое изложение

Задание

Редактор текста Vi

Вступление

Редактор Командной Строки

`vi` открытие файла

Режим ввода

Сохранение и выход

Другие способы просмотра файлов

Команда `cat`

Команда `less`

Навигация файлов в Vi

`h j k l` - left, down, up, right

`^ $` - начало и конец строки

`nG` -- перемещение на n-ную строку

`G` - перемещение на самую последнюю строку

`w` - перемещение в начало следующего слова

`nw` - перемещение вперед на n-ное количество слов

`b` - перемещение назад на предыдущее слово

`nb` - перемещение назад на n-ное количество слов

`{ }` - перемещение вперед/назад на один параграф

Удаление контента

`x` - удалить единственный символ

`nX` - удалить n-ное количество символов

`dd` - удалить текущую строку

`dnw` - удалить n-ное количество слов вперед

`dnb` - удалить n-ное количество слов назад

Отмена действия

Идём дальше

Итог

Важные концепции

Задания

Подстановочный знак Linux / Unix

Вступление

Как они выглядят?

`*` Подстановочный знак

`?` Подстановочный знак

`[]` Подстановочный знак

`^` - Нет

Примеры из реальной жизни

Задания

Права

Вступление

Права в Linux

`r` - право просмотра (read)

Смена прав

Сокращенный вид записи установки прав

Права для директорий

`root` пользователь

Задание

Фильтры

- Вступление
- Head
- tail
- sort
- nl
- wc
- cut
- sed
- uniq
- tac
- Others
- Краткое изложение
- Задания

Grep и регулярные выражения!

- Регулярное выражение
- egrep
- Обзор регулярных выражений
- Примеры

Конвейеризация и перенаправление

- Вступление
- Перенаправление в файл
- Сохранение в существующий файл
- Перенаправление из файла
- Перенаправить поток STDERR
- Конвейеризация
- Краткое содержание
- Задание

Управление процессами

- Вступление
- `top` Что работает в данный момент?
- `ps` процесс
- `kill` Убийство разбившегося процесса
- Задания переднего плана и фона
- Краткое содержание
- Задание.

Общие правила

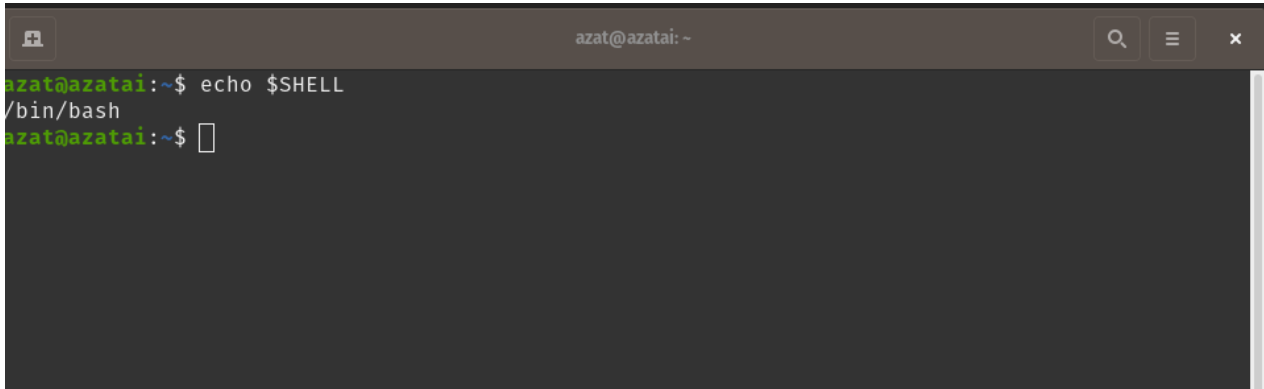
- В следующих страницах, я всякий раз буду ссылаться на Linux, предположим, что я на самом деле говорю Unix/ Linux. Linux является ответвлением Unix и ведет себя примерно так же.
- Всякий раз , когда вы видите "[]", это означает, что вы должны заменить это чем-то полезным. Замените все это (включая < and >). Если вы видите что-то подобное, то это обычно означает заменить это число.
- Всякий раз, когда вы видите **[something]**, это означает, что это является

необязательным. Когда вы выполняете команду, вы можете вставить что-то или оставить поле пустым.

Быстрый старт

`echo` для отображения системной переменной

```
echo $SHELL
```

A screenshot of a terminal window with a dark background. The title bar at the top shows 'azat@azatai: ~' and some window control icons. The terminal content shows a prompt 'azat@azatai:~\$' followed by the command 'echo \$SHELL'. The output is '/bin/bash'. The prompt then changes to 'azat@azatai:~\$' with a cursor.

Это показывает, что в настоящее время я использую `bash` в качестве оболочки по умолчанию.

Базовая навигация

`pwd` -- это сообщает вам, о текущем рабочем каталоге.

```
azat@azatai:~$ pwd
/home/azat
```

`ls` -- затем мы захотим узнать, что там есть

```
azat@azatai:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
azat@azatai:~$
```

`ls` это короткий **список**.

`**ls [параметры] [местоположение]**`

(`[]`) означает, что эти пункты являются необязательными.

мы можем выполнить команду с ними или без них.

```

azat@azatai:~$ ls -l
total 32
drwxr-xr-x 2 azat azat 4096 Jan 4 14:49 Desktop
drwxr-xr-x 3 azat azat 4096 Jan 4 19:22 Documents
drwxr-xr-x 2 azat azat 4096 Jan 4 19:55 Downloads
drwxr-xr-x 2 azat azat 4096 Jan 4 14:49 Music
drwxr-xr-x 2 azat azat 4096 Jan 4 20:07 Pictures
drwxr-xr-x 2 azat azat 4096 Jan 4 14:49 Public
drwxr-xr-x 2 azat azat 4096 Jan 4 14:49 Templates
drwxr-xr-x 2 azat azat 4096 Jan 4 14:49 Videos
azat@azatai:~$

```

Мы запустили **ls** с одним параметром командной строки (**-l**), который указывает, что мы собираемся сделать длинный список. Длинный список имеет следующее:

- Первый символ указывает, является ли это обычный файл (-) или каталог (d)
- Следующие 9 символов-это разрешения для файла или каталога (подробнее о них мы узнаем в разделе раздел 6).
- Следующее поле-это количество блоков (не беспокойтесь об этом слишком сильно).
- Следующее поле-владелец файла или каталога
- Следующее поле-группа, к которой принадлежит файл или каталог (в данном случае пользователи).
- Далее следует размер файла.
- Далее идет время модификации файла.
- Наконец, у нас есть фактическое имя файла или каталога

```

azat@azatai:~$ ls /home
azat  lost+found
azat@azatai:~$

```

Мы запустили **ls** с аргументом командной строки (/home). Когда мы делаем это, **ls** не перечисляет наши текущий каталог, но вместо этого перечислит содержимое этих каталогов.

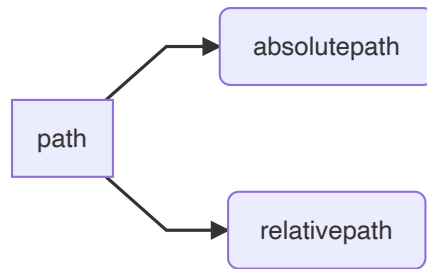
```

azat@azatai:~$ ls -l /home
total 20
drwxr-xr-x 19 azat azat 4096 Jan 4 19:12 azat
drwx----- 2 root root 16384 Jan 4 14:29 lost+found
azat@azatai:~$

```

Путь

В предыдущих командах мы начали касаться того, что называется путем. Я хотел бы более подробно остановиться на них сейчас, поскольку они важны для того, чтобы быть опытным в Linux. Всякий раз, когда мы ссылаемся на файл или каталог в командной строке, мы фактически ссылаемся на путь т.е. Путь - это способ добраться до определенного файла или каталога в системе.



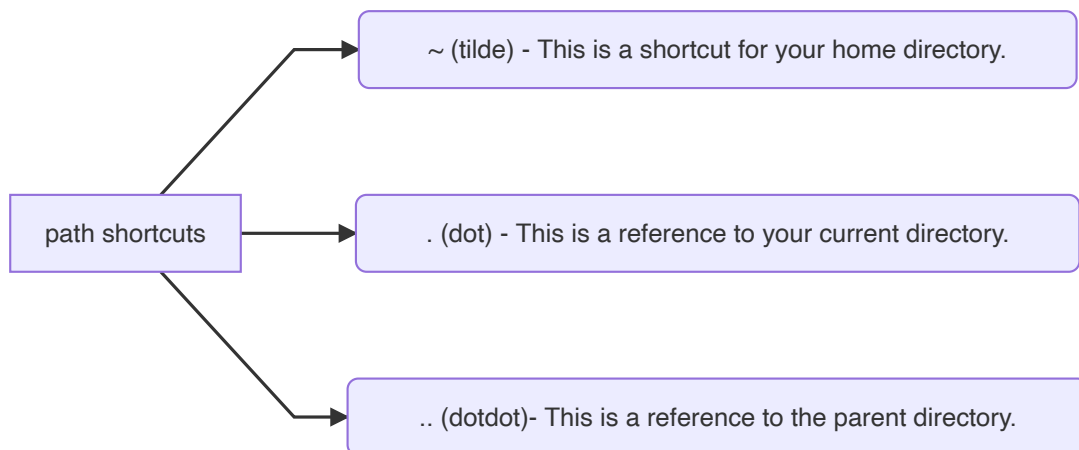
Есть 2 типа путей, которые мы можем использовать, абсолютные и относительные. Всякий раз, когда мы ссылаемся на файл или каталог, мы используем один из этих путей.

Файловая система под Linux представляет собой иерархическую структуру. В самом верху структуры находится так называемый корневой каталог. Он обозначается одной косой чертой (/). У него есть подкаталоги и у них тоже есть подкаталоги и так далее.

```
azat@azatai:~$ ls /  
bin  dev  home  lib32  libx32  media  opt  root  sbin  sys  usr  
boot  etc  lib  lib64  lost+found  mnt  proc  run  srv  tmp  var  
azat@azatai:~$
```

Абсолютные пути указывают расположение (файл или каталог) относительно корневого каталога. Вы можете легко их идентифицировать, так как они всегда начинаются с прямой косой черты (/)

Относительные пути определяют местоположение (файл или каталог) относительно того, где мы в данный момент находимся в системе. Они не начнут с косой черты. (Во многих случаях нам нравится использовать `.` в относительных путях)



```
azat@azatai:~$ pwd  
/home/azat  
azat@azatai:~$ ls ~  
Desktop  Downloads  Music      Public     Videos  
Documents Edraw      Pictures   Templates  
azat@azatai:~$ ls /home/azat
```

```

Desktop    Downloads  Music      Public     Videos
Documents  Edraw      Pictures   Templates

azat@azatai:~$ ls ~/Documents/
Note
azat@azatai:~$ ls /home/azat/Documents/
Note
azat@azatai:~$ ls ../../
bin  dev  home  lib32  libx32  media  opt  root  sbin  sys  usr
boot etc  lib  lib64  lost+found  mnt  proc  run  srv  tmp  var
azat@azatai:~$ ls /
bin  dev  home  lib32  libx32  media  opt  root  sbin  sys  usr
boot etc  lib  lib64  lost+found  mnt  proc  run  srv  tmp  var

```

cd Смена каталогов - перейти в другой каталог.

Синтаксис: ****cd [расположение]****

Если вы запустите команду **cd** без каких-либо аргументов, то она всегда приведет вас обратно в ваш домашний каталог.

```

azat@azatai:~$ pwd
/home/azat
azat@azatai:~$ cd Documents/
azat@azatai:~/Documents$ ls
Note
azat@azatai:~/Documents$ cd /
azat@azatai:/$ pwd
/
azat@azatai:/$ cd ~/Documents/
azat@azatai:~/Documents$ ls
Note
azat@azatai:~/Documents$ cd ../../
azat@azatai:/home$ ls
azat  lost+found
azat@azatai:/home$ cd .
azat@azatai:/home$ ls
azat  lost+found
azat@azatai:/home$

```

Завершение Вкладки

Командная строка имеет хороший маленький механизм, чтобы помочь нам в этом отношении. Это называется **Завершение табуляции**.

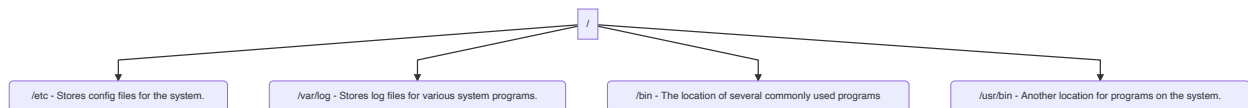
Когда вы начинаете вводить путь (в любом месте командной строки, вы не ограничены только определенными командами), вы можете нажать клавишу Tab на клавиатуре в любое время, что вызовет действие автоматического завершения.

попробовать это:

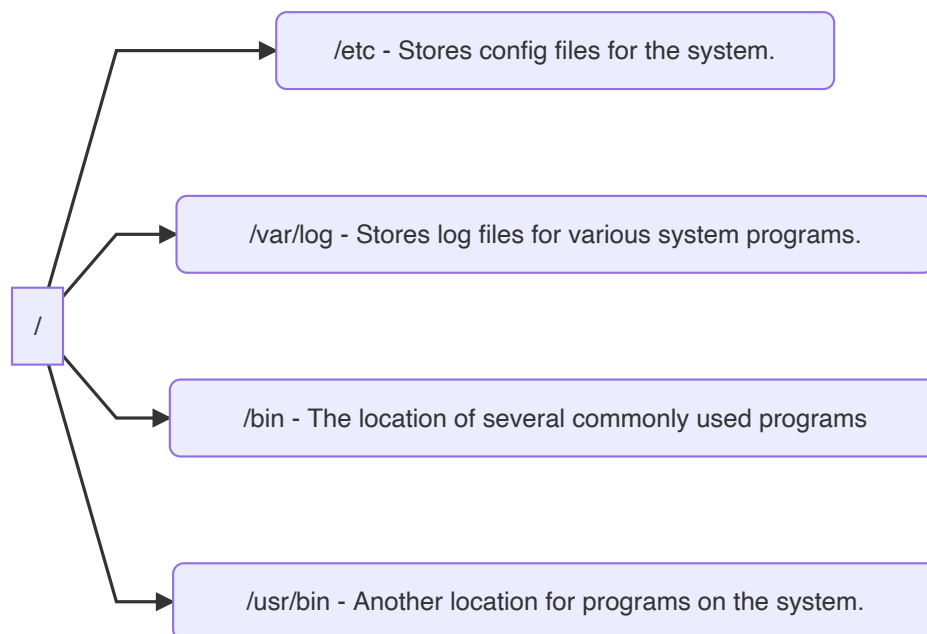
```
cd /hTab/<beginning of your username>Tab
```

Задания

- Напишите дерево каталогов (path tree) в вашей операционной системе (желательно, чтобы вы использовали Unix / Linux, как OS, или Mac OS)
- Что это за места в ваших хранилищах ОС? (Возьмем хотя бы 4 примера)
- Теперь перейдите в свой домашний каталог, используя 4 различных места и методы.
- Убедитесь, что вы используете табуляцию завершения при вводе ваших путей.



просто на случай, если вы не можете ясно видеть диаграмму:



Файлы

Все - это файл.

Итак, первое, что мы должны оценить в Linux, это то, что под капотом все на самом деле файл. Текстовый файл - это файл, каталог - это файл, ваша клавиатура - это файл (тот, который система только читает), ваш монитор - это файл (тот, в который система только записывает) и т. д. Начнем с того, что это не слишком повлияет на то, что мы делаем, но имейте это в виду, поскольку это помогает понять поведение Linux при управлении файлами и каталогами.

Linux - это система без расширения

Это может иногда быть трудно получить вашу голову вокруг, но как вы работаете через разделы, это начнет иметь больше смысла. Расширение файла обычно представляет собой набор из 2-4 символов после полной остановки в конце файла, который обозначает тип файла. Ниже приведены распространенные расширения:

- file.exe - исполняемый файл или программа. (Windows)
- file.txt - обычный текстовый файл.
- file.png, file.gif, file.jpg - изображение.
- file.azt - файл исходного кода программы azt. (Может быть прочитан и записан с помощью команды azt).
- file.aztx - исполняемый файл программы azt.
- file.pyzt - azt модуль python wrapper, может быть импортирован в python и использоваться непосредственно.
- file.jszt - azt модуль python wrapper, может быть импортирован в python и использоваться непосредственно.
- file.czt - оболочка azt модуля с / c++, может быть импортирована в javascript и использована непосредственно.
- file.azd - azt язык программирования сериализованный файл данных, вы можете сжать любой объект или любой вид данных .формат azd, если хотите.

В других системах, таких как Windows, расширение важно, и система использует его, чтобы определить, какой тип этого файла. В Linux система фактически игнорирует расширение и заглядывает внутрь файла, чтобы определить, какой это тип файла.

Так, например, у меня самого может быть файл.png, который является моей фотографией. Я мог бы переименовать файл на себя.txt или просто я и Linux по-прежнему с удовольствием рассматривали бы файл как файл изображения. Таким образом, иногда бывает трудно точно знать, какой тип файла является конкретным файлом. К счастью, есть команда под названием file, которую мы можем использовать, чтобы узнать это.

file [path] - получить информацию о том, какой тип файла файл или каталог.

```
azat@azatai:~$ ls .
azat.txt  Desktop    Downloads  Music      Public     Videos
azt       Documents  Edraw      Pictures   Templates
azat@azatai:~$ file azat.txt
azat.txt: ASCII text
azat@azatai:~$ file azt/
azt/: directory
```

Всякий раз, когда мы указываем файл или каталог в командной строке, это фактически путь. Кроме того, поскольку каталоги (как уже упоминалось выше) на самом деле являются просто особым типом файла, точнее сказать, что путь - это средство добраться до определенного места в системе, и это место - файл.

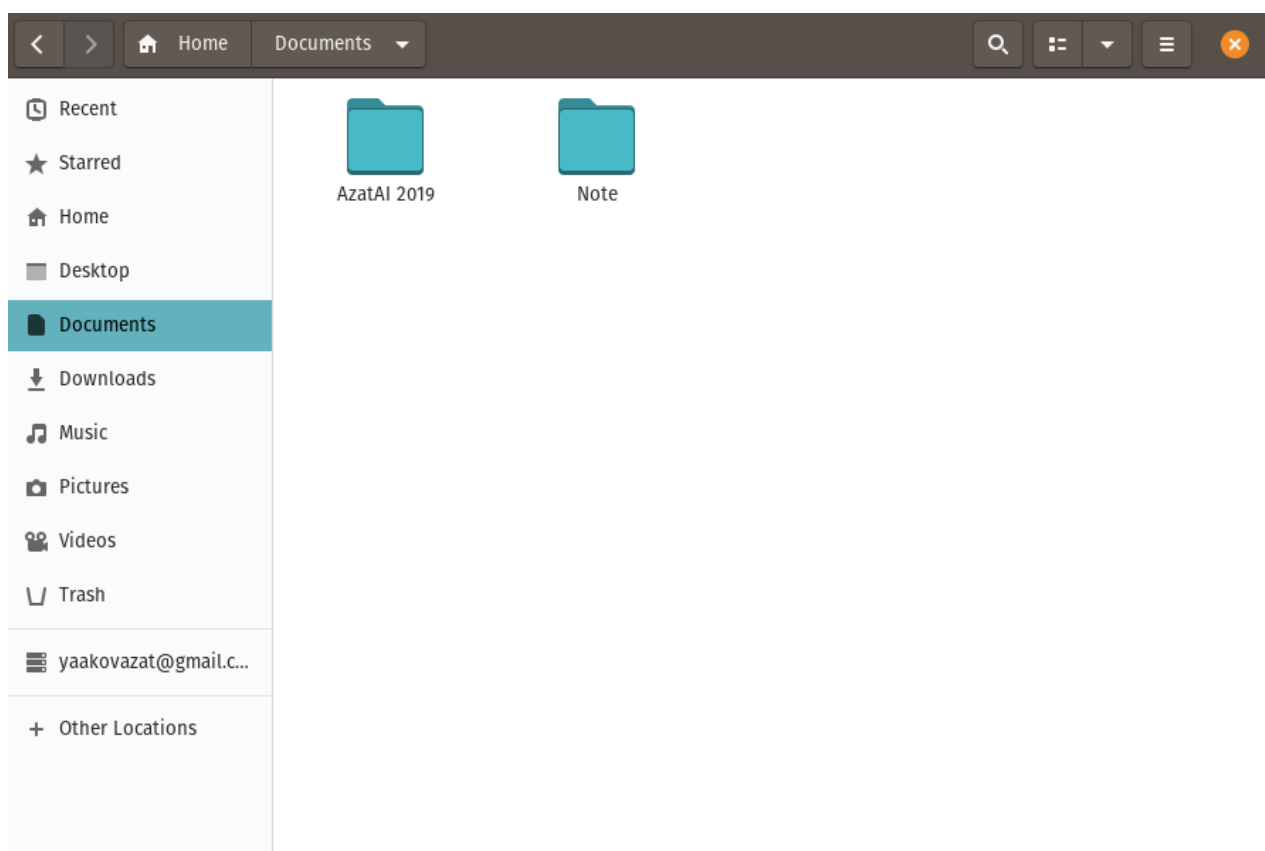
Linux чувствителен к регистру символов

```
azat@azatai:~$ cd documents
bash: cd: documents: No such file or directory
azat@azatai:~$ cd Documents
azat@azatai:~/Documents$ pwd
/home/azat/Documents
```

Очень важно знать, что, **Linux чувствителен к регистру**, в то время как другие операционные системы, такие как **Windows и Mac OS нечувствительны к регистру**.

Spaces in names

Хорошо, знайте, что у нас есть небольшая проблема, которую нужно решить сейчас, я только что создал папку (или каталог), кстати с именем `AzatAI 2019` и когда я хочу войти в каталог с `cd` что-то происходит:



```
azat@azatai:~/Documents$ pwd
/home/azat/Documents
azat@azatai:~/Documents$ cd AzatAI 2019
bash: cd: too many arguments
azat@azatai:~/Documents$
```

Что только произошло?

Как вы помните, пробел в командной строке - это способ разделения элементов. Так какую же дозу считает компьютер :

```
cd AzataI(the folder to move) 2019(another command)
```

и мы знаем, что `cd` получает путь только в качестве одного аргумента, и именно поэтому это произошло.

Есть два способа сделать это, и любой из них так же правомерен.

- S1: используя кавычки как `' '` или `" "`

```
azat@azatai:~/Documents$ cd 'AzataI 2019'
azat@azatai:~/Documents/AzataI 2019$ pwd
/home/azat/Documents/AzataI 2019
azat@azatai:~/Documents/AzataI 2019$ cd ..
azat@azatai:~/Documents$ cd "AzataI 2019"
azat@azatai:~/Documents/AzataI 2019$ pwd
/home/azat/Documents/AzataI 2019
```

- S2: использование escape-символов(`\`).

```
azat@azatai:~/Documents$ cd AzataI\ 2019
azat@azatai:~/Documents/AzataI 2019$ pwd
/home/azat/Documents/AzataI 2019
azat@azatai:~/Documents/AzataI 2019$
```

Другой метод заключается в использовании так называемого **escape - символа**, который представляет собой обратную косую черту (`\`). Обратная косая черта ускользает (или сводит на нет) от особого значения следующего символа.

В предыдущем разделе мы узнали о том, что называется завершением табуляции. Если вы используете его до того, как встретите пробел в имени каталога, то терминал автоматически экранирует любые пробелы в имени для вас.

```
cd AzataI\ 2019/
```

Скрытые файлы и каталоги.

Если имя файла или каталога начинается с точки `.` (полная остановка) тогда он считается скрытым в Unix/Linux. Чтобы сделать файл или каталог скрытым, все, что вам нужно сделать, это создать файл или каталог с именем, начинающимся с точки `.` или переименовать его, чтобы быть таковым. Команда **ls**, которую мы видели в предыдущем разделе, по умолчанию не выводит список скрытых файлов и каталогов. Мы можем изменить его, включив опцию командной строки **-a**, чтобы он показывал скрытые файлы и каталоги.

ls -a Перечислит содержимое каталога, включая скрытые файлы.

```
azat@azatai:~/Documents/AzatAI 2019$ ls
azat.ai
azat@azatai:~/Documents/AzatAI 2019$ ls -a
.  ..  .azatai  azat.ai
azat@azatai:~/Documents/AzatAI 2019$
```

или просмотрите короткое видео, чтобы вас поняли:

```
azat@azatai:ls
azatai
azat@azatai:ls -a
.azatai azat.ai
azat@azatai:
```

Краткое содержание

- Все является файлом в Linux

Даже справочники.

- Linux - это система без расширения

Файлы могут иметь любое расширение, которое им нравится, или вообще не иметь его.

- Linux чувствителен к регистру символов

Остерегайтесь глупых опечаток.

Задание

Хорошо, теперь давайте применим это на практике. Попробуйте следующее:

- Попробуйте запустить командный **file**, указав в нем несколько различных записей. Убедитесь, что при этом вы используете различные абсолютные и относительные пути.
- Теперь выполните команду, которая выведет список содержимого вашего домашнего каталога, включая скрытые файлы и каталоги.

Справочная страница

Вступление

Знаешь что, вспоминать все, что у тебя есть в голове, кажется крутым, но очень трудным, по крайней мере для меня. Linux представляет слишком много команд, что мы можем забыть их большую часть. Тем не менее, компьютер может запоминать и отмечать многое, поэтому, к счастью, есть способ помочь вам, назовём его ручными страницами или человеком для краткости.

Каковы они?

Ладно, для краткости это просто документирование команд. Круто, да? Что же они тогда делают? Как правило, они выполняют следующие действия для каждой команды, доступной в вашей системе:

1. Что они делают.
2. Как вы её можете использовать.
3. Какие аргументы имеются в командной строке.

Вызвать страницу справочника с помощью следующей команды:

```
man <command to lookup>
```

Например,

```
azat@azatai:~$ man ls
```

```
LS(1)                                User Commands                                LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is speci-
    fied.

    Mandatory arguments to long options are mandatory for short options
    too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        Manual page ls(1) line 1 (press h for help or q to quit)
```

NAME - что это за команда. (краткое описание команды)

SYNOPSIS - как должна выполняться эта команда. `[]` означает, что это необязательно.

DESCRIPTION - более подробное описание команды. Ниже описания всегда будет список всех параметров командной строки, доступных для данной команды.

Чтобы выйти из `man pages`, нажмите клавишу "q" для выхода.

Поисковик

Вы знаете, что хотите, но не знаете, какую команду именно использовать? Ищите это! Существует два способа поиска по руководствам.

1. Поиск по всем руководствам.

```
man -k <search term>
```

Например:

```
azat@azatai:~/Documents/AzatAI 2019$ man -k list
IO::Async::Listener (3pm) - listen on network sockets for incoming
connections
acl (5) - Access Control Lists
acpi_listen (8) - ACPI event listener
add-apt-repository (1) - Adds a repository into the /etc/apt/sources.list
or...
add-shell (8) - add shells to the list of valid login shells
Algorithm::Diff (3pm) - Compute `intelligent' differences between two
files ...
Algorithm::DiffOld (3pm) - Compute `intelligent' differences between two
fil...
american-english (5) - a list of English words
appres (1) - list X application resource database
apt-add-repository (1) - Adds a repository into the /etc/apt/sources.list
or...
argz (3) - functions to handle an argz list
argz_add (3) - functions to handle an argz list
argz_add_sep (3) - functions to handle an argz list
argz_append (3) - functions to handle an argz list
argz_count (3) - functions to handle an argz list
argz_create (3) - functions to handle an argz list
argz_create_sep (3) - functions to handle an argz list
argz_delete (3) - functions to handle an argz list
argz_extract (3) - functions to handle an argz list
```

Это позволит вам выполнить поиск по введенному вами ключевому слову (или вопросу запроса) из всех доступных руководств и распечатать результаты в алфавитном порядке.

2. Поиск в руководстве пользователя.

Нажмите сперва слэш `/` в пределах страницы справочника и введите ключевое слово (запрос), которое вы хотите найти:

```
ls - list directory contents

SYNOPSIS
ls [OPTION]... [FILE]...

DESCRIPTION
List information about the FILES (the current directory by
default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is
speci-
fied.

Mandatory arguments to long options are mandatory for short
options
too.

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author

/list
```

Long и Short варианты

Есть два различных варианта для нас, как вы можете найти выше, один начинается с `-` называется short hand options, а другой называется long hand options, который начинается с `--`. Сокращенный вариант `-a` делает то же самое, что и длинный вариант `--all`. Длинные варианты гораздо более удобочитаемы и легко запоминаются, в то время как короткие легко набираются.

Краткое содержание

```
man <command>
```

Найдите на странице руководства конкретную команду

```
man -k <search term>
```

Выполните поиск по ключевым словам на всех справочных страниц, содержащих данный поисковый запрос.

```
/<term>
```

На странице справочника выполните поиск по 'term'

n

После выполнения поиска на странице справочника выберите следующий найденный элемент.

Задание

1. Посмотрите на главную страницу с команды `cd`.
2. Поиск "change" с помощью различных двух способов из man-page(s).

Манипуляция файлами

`mkdir` Создание каталога

Linux организует свою файловую систему иерархическим образом. И как вы помните, все это файл в Linux. Мы создаем папки, особенно для структурирования и упорядочивания хранящихся в них данных. Выработайте привычку организовывать свои материалы в элегантную файловую структуру прямо сейчас, и вы будете благодарить себя в течение многих лет.

Создать каталог довольно просто, для этого у нас есть команда `mkdir`.

```
mkdir [options] <Directory>
```

```
~/Temp pwd
/Users/azat/Temp
~/Temp ls
~/Temp mkdir temp
~/Temp ls
temp
```

☐ Обновление до динамического gif, а не статического изображения.

```
~/Temp ls
first.cast temp temp0 temp2 tempdirectory
~/Temp exit
```

Однако, чтобы лучше понять, я думаю, что теперь вы можете попробовать выполнить эти команды и посмотреть, что произошло?


```
mkdir /home/azat/foo
mkdir ./blah
mkdir ../dir1
mkdir ~/linuxtutorialwork/dir2
```

Есть несколько полезных опций, доступных для **mkdir**. Мы можем легко найти их, просмотрев руководство `mkdir`.

```

MKDIR(1)                                BSD General Commands Manual                                MKDIR(1)

NAME
    mkdir -- make directories

SYNOPSIS
    mkdir [-pv] [-m mode] directory_name ...

DESCRIPTION
    The mkdir utility creates the directories named as operands, in the order
    specified, using mode
    rwxrwxrwx (0777) as modified by the current umask(2).

    The options are as follows:

    -m mode
        Set the file permission bits of the final created directory to
        the specified mode. The
        mode argument can be in any of the formats specified to the
        chmod(1) command. If a
        symbolic mode is specified, the operation characters '+' and
        '-' are interpreted
        relative to an initial mode of 'a=rwx'.
    :
```

Среди них два популярных используются часто:

mkdir [-p] при необходимости создайте родительские каталоги

Соблюдайте код ниже:

```
azat@pop-os:~/Temp$ ls
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ mkdir foo/foo2
mkdir: cannot creat directory 'foo/foo2':NO such file or directory
azat@pop-os:~/Temp$ mkdir -pv foo/foo2
mkdir: created directory 'foo'
mkdir: created directory 'foo/foo2'
azat@pop-os:~/Temp$
```

```
~/Temp mkdir temp/foo
mkdir: temp: No such file or directory
```

Нет такого файла или каталога

Почему?

Если мы создадим подкаталог с `mkdir`, `mkdir` ожидает, что у нас уже есть этот родительский каталог, и если нет, то появится ошибка.

Способ решить эту проблему заключается в использовании короткого варианта опции `-p`, или длинный вариант опции `--parents` который создаст для нас все необходимые родительские папки.

`-p`, `--p` stands for parents.

`mkdir [-v]` печать сообщения для каждого созданного каталога.

`-v` или `--verbose` докладывает после каждого созданного каталога.

```
azat@pop-os:~/Temp$ ls
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ mkdir foo/foo2
mkdir: cannot creat directory 'foo/foo2':NO such file or directory
azat@pop-os:~/Temp$ mkdir -pv foo/foo2
mkdir: created directory 'foo'
mkdir: created directory 'foo/foo2'
azat@pop-os:~/Temp$
```

Еще одна вещь, которую вы должны заметить здесь, это то, что мы можем писать различные варианты коротких опции вместе после `-`.

`rmdir` Удаление каталога.

Команда для удаления каталога `rmdir`, сокращенно от remove directory.

Вы должны быть осторожны, прежде чем сделать это, вы не сможете отменить это действие!!!

```
rmdir [options] <Directory>
```

Нужно отметить две вещи. Во-первых, `rmdir` поддерживает опции “-v” и “-p”, аналогичные `mkdir`. Во-вторых, каталог должен быть пустым, прежде чем его можно будет удалить (позже мы увидим способ обойти это).

```
azat@pop-os:~/Temp$ ls
foo
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
foo
azat@pop-os:~/Temp$ rmdir foo
rmdir: failed to remove 'foo': Directory not empty
azat@pop-os:~/Temp$
```

touch Создание пустого файла

Если мы коснемся файла, а он не существует, команда сделает нам одолжение и автоматически создаст его для нас. На самом деле прикосновение не предназначено для создания нового файла, но мы касаемся несуществующего файла, прикосновение создает его.

Что такое `touch` на самом деле: если мы касаемся файла, а он не существует, команда сделает нам одолжение и автоматически создаст его для нас.

```
azat@pop-os:~/Temp$ ls
foo
azat@pop-os:~/Temp$ touch test.azt
azat@pop-os:~/Temp$ ls
foo test.azt
azat@pop-os:~/Temp$
```

cp Копирование файла или каталога

Часто, прежде чем что-то изменить, мы можем захотеть создать дубликат, чтобы, если что-то пойдет не так, мы могли легко вернуться к оригиналу. Команда, которую мы используем для этого, `cp` что означает копирование.

```
cp [options] <source> <destination>
```

Обратите внимание, что и источник, и назначение являются путями. Это означает, что мы можем ссылаться на них, используя как абсолютные, так и относительные пути.

Используя опцию **-r**, которая расшифровывается как рекурсивная, мы можем копировать каталоги. Рекурсивность означает, что мы хотим посмотреть на каталог и все файлы и каталоги в нем, а для подкаталогов, войти в них и сделать то же самое и продолжать делать это.

```
azat@pop-os:~/Temp$ ls
foo test.azt
azat@pop-os:~/Temp$ cp foo -r foo2
azat@pop-os:~/Temp$ ls
foo foo2 test.azt
azat@pop-os:~/Temp$
```

mv Перемещение файла или каталога

Чтобы переместить файл, то используйте команду **mv**, который является коротким для перемещения. Он работает аналогично **cp**. Одно небольшое преимущество заключается в том, что мы можем перемещать каталоги без необходимости предоставлять опцию **-r**.

```
mv [options] <source> <destination>
```

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
foo foo2 test.azt
azat@pop-os:~/Temp$ mkdir backups
azat@pop-os:~/Temp$ mv foo2 ./backups/foo3
azat@pop-os:~/Temp$ ls
backups foo test.azt
azat@pop-os:~/Temp$ cd backups/
azat@pop-os:~/Temp/backups$ ls
foo3
azat@pop-os:~/Temp/backups$
```

mv Переименование файлов и каталогов

Обычно **mv** используется для перемещения файла или каталога в новый каталог, обратите внимание, что мы можем предоставить новое имя для файла или каталога, и в рамках перемещения он также переименует его.

В программировании, на самом деле, вы увидите много такого волшебства...

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
backups foo test.azt
azat@pop-os:~/Temp$ mv test.azt renamed.azt
azat@pop-os:~/Temp$ ls
backups foo renamed.azt
azat@pop-os:~/Temp$
```

rm Удаление файла (и непустых каталогов)

Команда для удаления или удаления файла является **rm**, что означает удалить.

```
rm [options] <file>
```

Попробуй:

```
mkdir foofoo
ls
rm foofoo
```

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
backups foo renamed.azt
azat@pop-os:~/Temp$ mkdir foofoo
azat@pop-os:~/Temp$ ls
backups foo foofoo renamed.azt
azat@pop-os:~/Temp$ rm foofoo/
rm: cannot remove 'foofoo/': Is a directory
azat@pop-os:~/Temp$
```

Не работа, верно? Тогда почему и как?

Почему: помните, что команда **rm** используется для удаления файла и непустых каталогов.

rm -r Удаление непустых каталогов

Как и некоторые другие команды, представленные в этом разделе, **rm** имеет несколько параметров, которые изменяют его поведение. Мы всегда увидим опции команды **rm** используя команду **man**:

```
man rm
```

Когда **rm** запускается с параметром **-r** он позволяет нам удалять каталоги и все файлы и каталоги, содержащиеся внутри.

OPTIONS

Remove (unlink) the FILE(s).

-f, --force

ignore nonexistent files and arguments, never prompt

-i prompt before every removal

-I prompt once before removing more than three files, or when removing recursively; less intrusive than **-i**, while still giving protection against most mistakes

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
backups clear foo foofoo ok renamed.azt
azat@pop-os:~/Temp$ rm -r backups/
azat@pop-os:~/Temp$ ls
clear foo foofoo ok renamed.azt
azat@pop-os:~/Temp$ rm -rf renamed.azt
azat@pop-os:~/Temp$ ls
clear foo foofoo ok
azat@pop-os:~/Temp$ rm -ri foo
rm: descend into directory 'foo'?
azat@pop-os:~/Temp$ ls
clear foo foofoo ok
azat@pop-os:~/Temp$ rm -ri foo
rm: descend into directory 'foo'? yes
rm: remove directory 'foo/foo2'? yes
rm: remove directory 'foo'? yes
azat@pop-os:~/Temp$ ls
clear foofoo ok
azat@pop-os:~/Temp$
```

Тут есть что объяснить:

1. **rm -r** удаляет каталоги и все файлы внутри каталога.
2. **rm -rf** путем добавления **-f** вариант, мы можем удалить что-либо силой, (никогда не подсказывать)

!!! Вы должны быть осторожны, чтобы сделать это, если вы новичок в Linux !!!

3. **rm -ri** это очень полезно, когда вы не совсем уверены, что вы удаляете.

Вы должны использовать `rm -ri` много, прежде чем вы достаточно опытни в Linux!

Краткое изложение

`mkdir`

Make Directory - ie. Create a directory.

`rmdir`

Remove Directory - ie. Delete a directory.

`touch`

Create a blank file.

`cp`

Copy - ie. Copy a file or directory.

`mv`

Move - ie. Move a file or directory (can also be used to rename).

`rm`

Remove - ie. Delete a file.

Задание

1. Вернитесь разными способами на домашний каталог.
2. Создайте каталог с именем `AzatAi`.
3. Имя каталога было неверным, теперь переименуйте его в `AzatAI`.
4. Теперь создайте два разных каталога внутри `AzatAI`, назовите их `foo` и `foofoo`
5. Создайте внутри `foo` новый файл `<your name>.txt` запишите туда ваши запланированные дела на сегодня в файл.
6. Скопируйте файл в `AzatAI/backups`.
7. Создайте файл `done.txt` внутри `foofoo`, и запишите задачи, которые вы закончили сегодня.
8. Резервное копирование всех `foo` и `foofoo` папки, чтобы переместить их в указанный адрес `Desktop/Backup0`.
9. Удалите все файлы и каталоги внутри программы `AzatAI` (в том числе `AzatAI`).

Примечание: Вы должны использовать интерактивный режим при удалении.

Редактор текста Vi

Вступление

Я полагаю, что вы использовали Microsoft Word, блокнот или, возможно, некоторые другие инструменты редактирования текста, такие как Atom, SublimeText, возможно даже Visual Studio Code. Все они являются хорошими текстовыми редакторами для использования в **GUI** (ГИП - графический интерфейс пользователя). Однако Vi - это текстовый редактор,

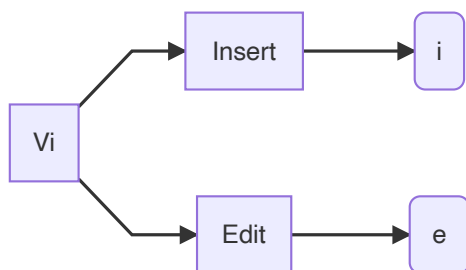
который, скорее всего, сильно отличается от любого другого редактора, который вы использовали раньше.

Vi - это очень мощный инструмент. Vi предустановлен почти во всех дистрибутивах Linux/Unix. Это мощный инструмент, используемый для создания, редактирования, обновления и поиска текстовых файлов в терминале. В основном сервера linux не имеют графический интерфейс пользователя (потому что он им не нужен), и вы можете использовать только текстовые редакторы доступные в терминале, вскоре вы поймете, насколько сильный и мощный Vi!

Редактор Командной Строки

Vi - это редактор командной строки, и все в Vi делается с помощью клавиатуры (вам не нужен ГИП, а также мышь).

В Vi есть два режима,



В режиме ввода вы можете вводить или удалять содержимое файла. В режиме редактирования вы можете перемещаться вокруг файла, выполнять такие действия, как удаление, копирование, поиск и замена, сохранение и т. д.

vi открытие файла

Когда мы запускаем vi, мы обычно выдаем его с одним аргументом командной строки, который является файлом, который вы хотели бы отредактировать. Также помните, что когда мы указываем файл, путь к нему может быть либо абсолютным либо относительным.

```
vi <file>
```

Теперь мы отредактируем наш первый файл.

При выполнении этой команды открывается файл. Если файл не существует, то он будет создан, затем вы открываете его. (нет необходимости "прикасаться", то есть использовать команду touch, перед их редактированием) после ввода vi он будет выглядеть что-то вроде этого (хотя в зависимости от того, в какой системе вы находитесь, может выглядеть немного иначе).


```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
clear foofoo ok
azat@pop-os:~/Temp$ vi first
```

Режим ввода

Вы всегда начинаете в режиме редактирования, поэтому первое, что мы сделаем, это переключимся в режим ввода нажав на **i**. Вы можете узнать, что вы находитесь в режиме ввода, взглянув на левый нижний угол.

```
insert mode now (press i to activate insert mode, and press ESC to exit insert mode)
```

```
azat@pop-os:~/Temp$ vi firstfile
```

Сохранение и выход

Обратите внимание: любая `command` которую вы вводите в Vi, должна быть введена в командном режиме (режим редактирования), не в режиме вставки, если это так, вы должны выйти из режима ввода, и перейти в режим редактирования нажав на **ESC**.

Существует несколько способов сохранить текущий файл в Vi:

- **zz** - сохранить и выйти
- **q!** - отменит все изменения (вы потеряете ваши последние изменения в файле), и выйдет. (Выход без сохранения)
- **:w** - сохранить
- **:wq** - сохранить и выйти

Любая команда, начинающаяся с двоеточия (:), требует, чтобы вы нажали, что-либо, чтобы завершить команду.

```
Intered the insert mode by typing i
```

```
Now let's save and exit.
```

```
azat@pop-os:~/Temp$ ls
clear foofoo ok
azat@pop-os:~/Temp$ vi firstfile
azat@pop-os:~/Temp$
```

Подумайте, в чем разница между `:w` и `:wq` ?

Другие способы просмотра файлов

Команда `cat`

`cat` это полезная команда в Linux (конечно все команды Linux полезны), она позволяет нам получить содержимое файла, не входя в файл.

```
cat <file>
```

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
clear firstfile foofoo ok
azat@pop-os:~/Temp$ cat firstfile
Intered the insert mode by typing i

Now let's save and exit.

azat@pop-os:~/Temp$
```

Эта команда хороша, когда у нас есть небольшой файл для просмотра, но если файл большой, то большая часть контента будет не влезать в экран и мы увидим только последнюю страницу контента. Для больших файлов нам лучше подходит команда **less**.

Команда `less`

`less` позволяет перемещаться вверх и вниз по файлу с помощью клавиш со стрелками. Вы можете пролистнуть всю страницу с помощью **пробела** или вернуться на страницу, нажав **b**. Когда вы закончите, вы можете нажать **q** для выхода.

```
less <file>
```

```
Intered the insert mode by typing i

Now let's save and exit.

firstfile (END)

azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
clear firstfile foofoo ok
azat@pop-os:~/Temp$ less firstfile
```

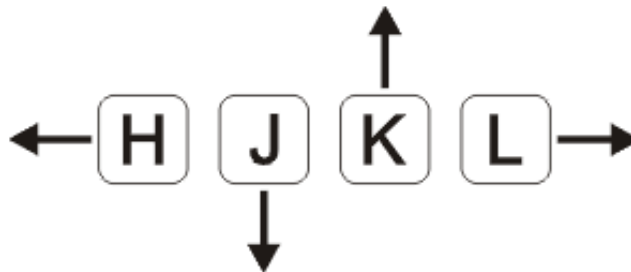
Навигация файлов в Vi

Ниже приведены некоторые из многих команд, которые вы можете использовать для перемещения по файлу. Поиграйте с ними и посмотрите, как они работают:

- **Клавиши со стрелками** - перемещает курсор вокруг
- **j,k,h,l** - двигает курсор вниз, вверх, влево и направо (работает также, как и клавиши со стрелками)
- **^** - переносит курсор в начало строки
- **\$** - переносит курсор в конец строки
- **nG** - переносит на **n-ную** строку (**5G** перенесет на пятую строку)
- **G** - переносит на последнюю строку
- **w** - переносит на следующее слово
- **nw** - переносит вперед на n-ное количество слов (2w переносит на два слова вперед)
- **b** - переносит на одно слово назад
- **nb** - переносит назад на n-ное количество слов назад
- **{** - следующий параграф
- **}** - предыдущий параграф

Вы должны будете практиковать все из этих команд, и запомнить все из них!!!!

Если вы наберете **:set nu** в режиме редактирования в **vi**, это позволит включить пронумеровать строки. С помощью данной функции, я нахожу поиск нужных строк намного легче.



```
azat@pop-os:~/Temp$ vi azt.md
azat@pop-os:~/Temp$
```

h j k l - left, down, up, right

Попробуйте перемещаться по файлу используя **h(left)j(down)k(up)l(right)**

^ \$ - начало и конец строки

Попробуйте переместиться в начало и конец строки используя **^** и **\$**

```
azat@pop-os:~/Temp$ vi azt.md
azat@pop-os:~/Temp$ vi azt.md
```

nG -- перемещение на n-ную строку

Попробуйте перемещаться по строкам используя **nG**

```
Project description
Intro for ~azt Programming language
~azt programming language
```

G - перемещение на самую последнюю строку

Переместись на последнюю строку используя:

Последняя строка: **G**

W - перемещение в начало следующего слова

nw - перемещение вперед на n-ное количество слов

b - перемещение назад на предыдущее слово

nb - перемещение назад на n-ное количество слов

{ } - перемещение вперед/назад на один параграф

Совет

Если вы наберете **:set nu** в режиме редактирования в vi, это позволит включить пронумеровать строки. С помощью данной функции, я нахожу поиск нужных строк намного легче.

Удаление контента

Ниже приведены некоторые из многих способов, которыми мы можем удалить содержимое в vi.

- **x** - удалить единственный символ
- **nx** - удалить n-ное количество символов (например, 5x удаляет пять символов)
- **dd** - удалить текущую строку
- **dn** - d команда перемещения. Команда для удаления содержимого после перемещения. (d5w означает что он удалит 5 слов)

x - удалить единственный символ

nX - удалить n-ное количество символов

dd - удалить текущую строку

dnw - удалить n-ное количество слов вперёд

dnb - удалить n-ное количество слов назад

Отмена действия

Отменить изменения в **vi** очень легко. За это отвечает **u**.

- **u** - отмена последнего действия (вы можете продолжать нажимать u для дальнейшей отмены)
- **U (С заглавной)** - отменяет все последние действия текущей строки

Идём дальше

Теперь мы можем вставлять содержимое в файл, перемещаться по нему, удалять содержимое, отменять действия, а затем сохранять и выходить. Вы уже изучили основные понятия и команды Vi. Однако, никогда не останавливайтесь изучать глубже и подробнее, если вы решили идти вперёд, пожалуйста, обратитесь к следующим ресурсам:

- [Основные команды в Vi](#)
- [Редактор Vi в UNIX](#)

Конечно, там много "cheet sheet", в которых ты можешь увидеть много команд.

Если это возможно, вам стоит изучить команды, которые приведены ниже, для того, чтобы повысить вашу эффективность.

- copy and paste
- search and replace
- buffers
- markers
- ranges
- settings

vi поначалу может быть сложным в изучении, но не сдавайтесь и продолжайте работать, вскоре он сможет стать вашим надежным другом.

Итог

Мы научились:

vi - Редактирование файла

cat - Просмотр содержимого файла

less - Удобная команда для просмотра большого количества данных в файле

Важные концепции

Без мышки

`vi` это редактор текстов, который работает идеально в терминале, и вам не нужен графический интерфейс для его использования. Чтобы стать квалифицированным разработчиком, работа в `vi` одна из самых первых навыков, которые вы должны освоить.

Команды редактирования

Есть много команд редактирования в `vi`, никогда не переставайте пробовать и учиться, помните, практика приводит к совершенству.

Задания

Давайте попрактикуемся:

- Создайте текстовый файл, и напишите в нём ваше резюме.
- Сохраните файл, и просмотрите его с помощью `cat` и `less`.
- Практикуйтесь в перемещении по файлу.
- Практикуйтесь в удалении информации.
- Отмените изменения и выйдите из файла.

Подстановочный знак Linux / Unix

Вступление

Подстановочный знак - это общий термин, относящийся к чему-то, что может быть заменено на все возможности.

В компьютерных терминах, обычно простой "подстановочный знак" - это просто `*`, который может соответствовать одному или нескольким символам, и, возможно, `?` это может соответствовать любому отдельному символу.

Если вы изучали регулярное выражение раньше, вы можете запутаться, потому что подстановочные знаки кажутся регулярными выражениями, однако регулярные выражения очень мощные и гораздо более сложные, чем подстановочные знаки, и мы догоним их позже.

Как они выглядят?

Вот основной набор подстановочных знаков:

- `*` - представляет ноль или более символов
- `?` - представляет собой один символ
- `[]` - представляет диапазон символов

`*` Подстановочный знак

`*` - представляет ноль или более символов

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
azt.md blahblah.txt clear foofoo ok.request
blafiii.txt bob.md firstfile ok
azat@pop-os:~/Temp$ ls b*
blafiii.txt blahblah.txt bob.md
azat@pop-os:~/Temp$
```

Как вы можете убедиться, **подстановочные знаки** очень полезны.

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
azt.md blahblah.txt clear foofoo ok.request
blafiii.txt bob.md firstfile ok
azat@pop-os:~/Temp$ ls b*
blafiii.txt blahblah.txt bob.md
azat@pop-os:~/Temp$ ls bl*
blafiii.txt blahblah.txt
azat@pop-os:~/Temp$
```

Как насчет еще нескольких примеров?

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
azt.md blahblah.txt clear foofoo ok.request
blafiii.txt bob.md firstfile ok
azat@pop-os:~/Temp$ ls b*
blafiii.txt blahblah.txt bob.md
azat@pop-os:~/Temp$ ls bl*
blafiii.txt blahblah.txt
azat@pop-os:~/Temp$ ls *.txt
blafiii.txt blahblah.txt
azat@pop-os:~/Temp$
```

```
ls *.txt
```

Этот код покажет нам все .txt-файл.

? Подстановочный знак

Теперь давайте представим **?** оператор. **?** - представляет собой **один символ**. В приведенном ниже примере мы ищем каждый файл, вторая буква которого - **l**.

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
Almaty blafiii.txt bob.md foofoo ok.request
azt.md blahblah.txt firstfile ok
azat@pop-os:~/Temp$ ls ?l*
Almaty blafiii.txt blahblah.txt
azat@pop-os:~/Temp$
```

Или как насчет каждого файла с расширением из трех букв?

```
azat@pop-os:~/Temp$ ls *.???
blafiii.txt blahblah.txt love.mp3 secret.mp4
azat@pop-os:~/Temp$
```

[] Подстановочный знак

В отличие от предыдущих 2-х подстановочных знаков, которые указывали любой символ, оператор диапазон позволяет ограничиться подмножеством символов. **[]** -представляет диапазон символов

В этом примере мы ищем каждый файл, имя которого начинается с буквы **s** или **v**.

```
azat@pop-os:~/Temp$ ls [vs]*
secret.mp4 video.mepg
azat@pop-os:~/Temp$
```

Так, например, если бы мы хотели найти каждый файл, имя которого содержит цифру, мы могли бы сделать следующий:

```
azat@pop-os:~/Temp$ ls *[0-9]*
file2.txt fool love.mp3 secret.mp4
azat@pop-os:~/Temp$
```

^ - Нет

Мы также можем изменить диапазон с помощью символа (^), что означает поиск любого символа, который не является один из следующих.

```
azat@pop-os:~/Temp$ ls [^a-k]*
Almaty love.mp3 ok.request secret.mp4 video.mepg

ok:
okk
azat@pop-os:~/Temp$
```

Примеры из реальной жизни

1. Найдите тип каждого файла в каталоге.

```
file ./*
```

```
azat@pop-os:~/Temp$ file ./*
./Almaty:          empty
./azt.md:          ASCII text, with very long lines
./blafiii.txt:     empty
./blahblah.txt:    empty
./bob.md:          empty
./file2.txt:       empty
./firstfile:       ASCII text
./fool:            empty
./foofoo:          directory
./love.mp3:        empty
./ok:              directory
./ok.request:      empty
./secret.mp4:      empty
./video.mepg:      empty
azat@pop-os:~/Temp$
```

2. Переместите все файлы типа jpg или png (файлы изображений) в другой каталог.

```
mv *.*??g ./images
```

```

azat@pop-os:~/Temp$ ls
Almaty          blafiii.txt    file2.txt      foofoo         love.mp3       secret.mp4
AzatAI_logo.png blahblah.txt    firstfile      images         ok             video.mpeg
azt.md          bob.md         fool           logo.jpg       ok.request
azat@pop-os:~/Temp$ mv *.*??g ./images/
azat@pop-os:~/Temp$ ls
Almaty          blahblah.txt    firstfile      images         ok.request
azt.md          bob.md         fool           love.mp3       secret.mp4
blafiii.txt     file2.txt       foofoo         ok             video.mpeg
azat@pop-os:~/Temp$ cd images/
azat@pop-os:~/Temp/images$ ls
AzatAI_logo.png logo.jpg
azat@pop-os:~/Temp$

```

3. Узнайте размер и время модификации.файл `.bash_history` в каждом доме пользователя каталог.

```
ls -lh /home/*/.bash_history
```

``.bash_history`` это файл в обычном домашнем каталоге пользователя, который хранит историю команд, введенных пользователем в командной строке. Вспомните, как это было ``.`` значит, это скрытый файл?

```

```bash
azat@pop-os:~/Temp$ ls -lh /home/*/.bash_history
-rw----- 1 azat azat 2.4K Jan 9 16%59 /home/azat/.bash_history
azat@pop-os:~/Temp$

```

## Задания

Давайте поиграем с некоторыми шаблонами.

- Хорошим каталогом для игры является каталог `/etc`, содержащий конфигурационные файлы для системы. Как обычный пользователь, вы можете просматривать файлы, но вы не можете вносить какие-либо изменения, поэтому мы не можем причинить никакого вреда. Сделайте список этого каталога, чтобы увидеть, что там есть. Затем выберите различные подмножества файлов и посмотрите, можно ли создать шаблон для выбора только этих файлов.
- Сделайте список `/etc` только с файлами, которые содержат расширение.
- А как насчет расширения только на 3 буквы?
- Как насчет файлов, имя которых содержит заглавную букву? (подсказка: `[:upper:]`)

здесь может пригодиться)

- Можете ли вы перечислить файлы, имя которых имеет длину 4 символа?

```
ls /etc/*.*
```

```
azat@pop-os:~/Temp$ ls -l /etc/*.*
```

```
ls -l /etc/*.???
```

```
azat@pop-os:~/Temp$ ls -l /etc/*.???
```

```
azat@pop-os:~/Temp$
```

```
ls -l /etc/*[[:upper:]]*
```

```
azat@pop-os:~/Temp$ ls -l /etc/*[[:upper:]]
```

```
azat@pop-os:~/Temp$
```

```
ls /etc/????
```

```
azat@pop-os:~/Temp$ ls ????
foo1
azat@pop-os:~/Temp$ clear
```

## Права

---

### Вступление

Права указывают, что конкретное лицо может или не может делать в отношении файла или каталога.

Таким образом, права важны для создания безопасной среды. К счастью, права в системе linux довольно просты в работе.

# Права в Linux

Для каждого файла мы определяем 3 набора людей, для которых мы можем указать разрешения.

- **r** read - вы можете увидеть контент файла
- **w** write - вы можете изменять содержимое файла
- **x** execute - Вы можете запускать программу или скрипт
- **owner** - одиночная персона владелец файла. (Как правило, тот, кто создал файл, но право собственности может быть предоставлено другим.)
- **group** - каждый файл принадлежит к одной группе.
- **others** - все остальные, кто не входит в группу или не является владельцем.

## **r** - право просмотра (read)

Чтобы просмотреть разрешения для файла, мы используем команду `ls -l`.

```
ls -l [path]
```

```
azat@pop-os:~/Temp$ ls -l ok
total 4
drwxr-xr-x 2 azat azat 4096 Jan 9 10:19 okk
azat@pop-os:~/Temp$
```

```
azat@pop-os:~/Temp$ ls -l ok
total 4
drwxr-xr-x 2 azat azat 4096 Jan 9 10:19 okk
```

- Первый символ определяет тип файла. Если это тире ( - ), то это обычный файл. Если это d тогда это каталог. (здесь `d` )
- Следующие 3 символа представляют права доступа для владельца. Буква обозначает наличие разрешения, а тире ( - ) - отсутствие разрешения. В этом примере владелец имеет все разрешения (чтение, запись и выполнение). ( здесь `rw` )
- Следующие 3 символа представляют права доступа для группы. В этом примере пользователь имеет возможность читать и запускать, но не писать.(здесь `r-x` )
- Наконец, последние 3 символа представляют права доступа для других (или всех остальных). (здесь `r-x` )

## Смена прав

Для изменения прав доступа к файлу или каталогу мы используем команду **chmod** которая расшифровывается как change file mode bits.

```
chmod [permissions] [path]
```

**chmod** имеет аргументы прав, которые состоят из 3 компонентов

- Для кого мы меняем разрешение? Пользователь? Владелец? Группа ?
- Мы даем ( + ) или отзывает ( - ) права?
- Какое разрешение мы устанавливаем? - чтение (r), запись (w) или выполнение (x)?

Перейдем к практике:

1. Для файла `main.azt`, предоставьте группе разрешение на выполнение.

```
azat@pop-os:~/Temp$ ls -l main.azt
-rw-r--r-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$ chmod g+x main.azt
azat@pop-os:~/Temp$ ls -l main.azt
-rw-r-xr-- 1 azat azat 0 Jan 9 21:15 main.azt
```

2. Затем удалите разрешение на запись для владельца.

```
azat@pop-os:~/Temp$ chmod u-w main.azt
azat@pop-os:~/Temp$ ls -l main.azt
-r--r-xr-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$
```

3. Назначить несколько разрешений одновременно:

```
azat@pop-os:~/Temp$ ls -l main.azt
-r--r-xr-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$ chmod gu+wx main.azt
azat@pop-os:~/Temp$ ls -l main.azt
-rwxrwxr-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$ chmod go-wx main.azt
azat@pop-os:~/Temp$
```

```
azat@pop-os:~/Temp$ ls -l main.azt
-r--r-xr-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$ chmod gu+wx main.azt
azat@pop-os:~/Temp$ ls -l main.azt
-rwxrwxr-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$ chmod go-wx main.azt
azat@pop-os:~/Temp$ ls -l main.azt
-rwxr--r-- 1 azat azat 0 Jan 9 21:15 main.azt
azat@pop-os:~/Temp$
```

## Сокращенный вид записи установки прав

Существует сокращенный метод работы, а также для указания разрешений. Наша типичная система счисления - десятичная. Это базовая система счисления 10 и как таковая имеет 10 символов (0-9). Другая система счисления - восьмеричная, которая является основанием 8 (0-7). Теперь просто так получается, что с 3 разрешениями и каждый из них включен или выключен, у нас есть 8 возможных комбинаций ( $2^3$ ). Теперь мы можем также представить числа, используя двоичный код, который имеет только 2 символа (0 и 1). Отображение восьмеричного числа в двоичное представлено в таблице ниже.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Теперь интересно отметить, что мы можем представить все 8 восьмеричных значений с 3 двоичными битами и что каждая возможная комбинация 1 и 0 включена в него. Таким образом, у нас есть 3 бита, и у нас также есть 3 разрешения.

Если вы думаете, что 1 представляет собой ВКЛ, а 0-ВЫКЛ, то одно восьмеричное число может использоваться для представления набора разрешений для набора пользователей.

Три числа, и мы можем указать разрешения для пользователя, группы и других. Давайте рассмотрим несколько примеров. (обратитесь к таблице выше, чтобы увидеть, как они совпадают)

```
azat@pop-os:~/Temp$ touch demo
azat@pop-os:~/Temp$ ls -l demo
-rw-r--r-- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 751 demo
azat@pop-os:~/Temp$ ls -l demo
-rwxr-x--x 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$
```

Люди часто запоминают часто используемые числовые последовательности для различных типов файлов и находят этот метод довольно удобным. Например, **755** или **750** обычно используются для скриптов.

Если вы запутались тогда, я хочу, чтобы это было немного ясно:

Мы используем три числа, верно ? Тогда это означает, что первое число - для пользователя, второе-для группы и третье - для других.

```
azat@pop-os:~/Temp$ ls -l demo
----- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 1 demo
azat@pop-os:~/Temp$ ls -l demo
-----x 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 2 demo
azat@pop-os:~/Temp$ ls -l demo
-----w- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 3 demo
azat@pop-os:~/Temp$ ls -l demo
-----wx 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 4 demo
azat@pop-os:~/Temp$ ls -l demo
-----r-- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 5 demo
azat@pop-os:~/Temp$ ls -l demo
-----r-x 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 6 demo
azat@pop-os:~/Temp$ ls -l demo
-----rw- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 7 demo
azat@pop-os:~/Temp$ ls -l demo
-----rwx 1 azat azat 0 Jan 9 21:30 demo
```

```
azat@pop-os:~/Temp$ ls -l
----- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 0 demo
azat@pop-os:~/Temp$ ls -l demo
----- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 1 demo
azat@pop-os:~/Temp$ ls -l demo
-----x 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 2 demo
azat@pop-os:~/Temp$ ls -l demo
-----w- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 3 demo
azat@pop-os:~/Temp$ ls -l demo
-----wx 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 4 demo
azat@pop-os:~/Temp$ ls -l demo
-----r-- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 5 demo
azat@pop-os:~/Temp$ ls -l demo
-----r-x 1 azat azat 0 Jan 9 21:30 demo
```

```

azat@pop-os:~/Temp$ chmod 6 demo
azat@pop-os:~/Temp$ ls -l demo
-----rw- 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$ chmod 7 demo
azat@pop-os:~/Temp$ ls -l demo
-----rwx 1 azat azat 0 Jan 9 21:30 demo
azat@pop-os:~/Temp$

```

Вы также должны быть осторожны с порядком разрешения.

	u g o		
	754		
	/   \		
access	r w x	r w x	r w x
binary	4 2 1	4 2 1	4 2 1
enabled	1 1 1	1 0 1	1 0 0
result	4 2 1	4 0 1	4 0 0
total	7	5	4

Используйте команду `chmod` для установки прав доступа к файлам.

Команда `chmod` использует в качестве аргумента трехзначный код.

Три цифры кода `chmod` устанавливают разрешения для этих групп в следующем порядке:

1. Владелец (вы)
2. Группа (группа других пользователей, которую вы настроили)
3. Мир (все кто еще просматривает файловую систему)

Каждая цифра этого кода устанавливает разрешения для одной из этих групп следующим образом. Читать - это 4. Писать - это 2. Выполнять - это 1.

Суммы этих чисел дают комбинации этих разрешений:

- 0 = нет никаких разрешений; этот человек не может читать, записывать или выполнять файл.
- 1 = только выполнение
- 2 = только запись
- 3 = запись и выполнение (1+2)



- 4 = только чтение
- 5 = чтение и выполнение (4+1)
- 6 = чтение и запись (4+2)
- 7 = чтение, запись и выполнение (4+2+1)

## Права для директорий

Один и тот же набор разрешений может использоваться для каталогов, но они имеют несколько иное поведение.

- **r** - у вас есть возможность читать содержимое каталога (т. е. использовать `ls`)
- **w** - у вас есть возможность записывать в каталог (т. е. создавать файлы и каталоги)
- **x** - у вас есть возможность войти в этот каталог (т. е. `cd`)

```
azat@pop-os:~/Temp$ ls test
file1 file2 file3
azat@pop-os:~/Temp$ chmod 400 test
azat@pop-os:~/Temp$ ls -ld test
dr----- 2 azat azat 4096 Jan 9 21:52 test
azat@pop-os:~/Temp$ cd test/
bash: cd: test/: Permission denied
azat@pop-os:~/Temp$ ls test
ls: cannot access 'test/file1': Permission denied
ls: cannot access 'test/file2': Permission denied
ls: cannot access 'test/file3': Permission denied
file1 file2 file3
azat@pop-os:~/Temp$ chmod 100 test
azat@pop-os:~/Temp$ ls test
ls: cannot open directory 'test': Permission denied
azat@pop-os:~/Temp$ cd test
azat@pop-os:~/Temp/test$ pwd
/home/azat/Temp/test
```

Обратите внимание, что выше в 4 строке, когда мы запускали `ls`, я добавил опцию `-d`, которая обозначает каталог. Обычно, если мы даем `ls` аргумент, который является каталогом, он будет перечислять содержимое этого каталога. В этом случае, мы заинтересованы в разрешениях каталога непосредственно, и опция `-d` позволяет нам получить это.

Эти разрешения могут показаться немного запутанными на первый взгляд. Мы должны помнить, что эти разрешения предназначены для самого каталога, а не для файлов внутри него. Так, например, у вас может быть каталог, для которого у вас нет разрешения на чтение. В нем могут быть файлы, для которых у вас есть разрешение на чтение. Пока вы знаете, что файл существует, и его имя, вы все еще можете прочитать файл.

### root пользователь

В системе Linux обычно есть только 2 человека, которые могут изменить разрешения файла или каталога. Владелец файла или каталога и пользователь `root`.

Пользователь root - это суперпользователь, которому разрешено делать все и вся в системе. Обычно администраторы системы являются единственными, кто имеет доступ к учетной записи root и использует ее для обслуживания системы. Чаще всего, обычные пользователи в основном имеют доступ только к файлам и каталогам в своем домашнем каталоге и, возможно, к нескольким другим для совместного использования и совместной работы, и это помогает поддерживать безопасность и стабильность системы.

## Задание

По практикуйтесь в изменении прав у файлов и каталогов

- Создайте папку с вашими личными статьями и изображениями.
- Установите права на все изображения с помощью широких карт, чтобы только вы могли читать файл.
- Задайте разрешение для файлов doc или docx, чтобы только вы и группа могли писать.
- Установите разрешение для своей папки, чтобы только вы могли войти в нее.

## Фильтры

С помощью фильтров мы можем легко превратить данные в информацию или легко извлечь информацию, которую мы хотим. Один из основополагающих принципов Linux заключается в том, что каждый элемент должен делать одну и только одну вещь, и что мы можем легко объединить эти элементы вместе.

## Вступление

Фильтр в контексте командной строки Linux - это программа, которая принимает текстовые данные и затем преобразует их определенным образом. Фильтры - это способ взять необработанные данные, созданные другой программой или сохраненные в файле, и манипулировать ими, чтобы они отображались более подходящим для нас способом.

## Head

Head это программа, которая печатает первые строки своего ввода. По умолчанию выводит 10 строк.

```
head [-number of lines to print] [path]
```

```
azat@pop-os:~/Temp$ head mysampleddata.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
```

```
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Bettylimes 14
azat@pop-os:~/Temp$ head mysampleddata.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
azat@pop-os:~/Temp$
```

Конечно, мы можем показать столько линий, сколько захотим:

```
azat@pop-os:~/Temp$ head -3 mysampleddata.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
azat@pop-os:~/Temp$ head 3 mysampleddata.txt
head: cannot open '3' for reading: No such file or directory
==> mysampleddata.txt <==
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
```

Обратите внимание, что там перед числом стоит дефис!

## tail

Tail - это противоположность head. Tail - это программа, которая печатает последние строки ввода.

```
tail [-number of lines to print] [path]
```

```
azat@pop-os:~/Temp$ tail -3 mysampleddata.txt
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
```

## sort

Sort будет сортировать его вход, красиво и просто. По умолчанию сортировка производится в алфавитном порядке, но для изменения механизма сортировки доступно множество опций.

```
sort [-options] [path]
```

```
azat@pop-os:~/Temp$ sort mysampleddata.txt
Anne mangoes 7
Betty limes 14
Fred apples 20
Greg pineapples 3
Lisa peaches 7
Mark grapes 39
Mark watermellons 12
Oliver rockmellons 2
Robert pears 4
Susy oranges 12
Susy oranges 5
Terry oranges 9
azat@pop-os:~/Temp$
```

## nl

`nl` означает нумерованные линии, и это именно так.

```
azat@pop-os:~/Temp$ sort mysampleddata.txt
1 Anne mangoes 7
2 Betty limes 14
3 Fred apples 20
4 Greg pineapples 3
5 Lisa peaches 7
6 Mark grapes 39
7 Mark watermellons 12
8 Oliver rockmellons 2
9 Robert pears 4
10 Susy oranges 12
11 Susy oranges 5
12 Terry oranges 9
```

```
azat@pop-os:~/Temp$ nl mysampleddata.txt
 1 Fred apples 20
 2 Susy oranges 5
 3 Mark watermellons 12
 4 Robert pears 4
 5 Terry oranges 9
 6 Lisa peaches 7
 7 Susy oranges 12
 8 Mark grapes 39
 9 Anne mangoes 7
10 Greg pineapples 3
11 Oliver rockmellons 2
12 Betty limes 14
```

Основное форматирование в порядке, но иногда вы ищете что-то немного другое. С несколькими параметрами командной строки, **nl** с радостью обяжет.

```
nl -s '. ' -w 10 mysampleddata.txt
```

В приведенном выше примере мы использовали 2 параметра командной строки. В первом из них указывается **-s**, что должно быть напечатано после номера, а во втором **-w**-сколько отступов нужно поставить перед номерами.

**-s, --number-separator=STRING**

добавить строку после (возможного) номера строки

**-w, --number-width=NUMBER**

использовать числовые столбцы для номеров строки

```
azat@pop-os:~/Temp$ nl -s '. ' -w 10 mysampleddata.txt
 1 Fred apples 20
 2 Susy oranges 5
 3 Mark watermellons 12
 4 Robert pears 4
 5 Terry oranges 9
 6 Lisa peaches 7
 7 Susy oranges 12
 8 Mark grapes 39
 9 Anne mangoes 7
10 Greg pineapples 3
11 Oliver rockmellons 2
12 Betty limes 14
azat@pop-os:~/Temp$
```

## WC

**wc** означает количество слов.

```
wc [-options] [path]
```

```
azat@pop-os:~/Temp$ wc mysampleddata.txt
12 36 197 mysampleddata.txt
```

Иногда вы просто хотите одну из этих ценностей. **-l** даст нам только строки, **-w** даст нам слова и **-m** даст нам символы.

```
azat@pop-os:~/Temp$ wc -l mysampleddata.txt
12 mysampleddata.txt
azat@pop-os:~/Temp$ wc -m mysampleddata.txt
197 mysampleddata.txt
azat@pop-os:~/Temp$ wc -w mysampleddata.txt
36 mysampleddata.txt
```

Вы можете объединить аргументы командной строки. Этот пример дает нам и строки, и слова.

```
azat@pop-os:~/Temp$ wc -lw mysampleddata.txt
12 36 mysampleddata.txt
```

## cut

cut это небольшая программа, которую можно использовать, если ваше содержимое разделено на поля (столбцы), и вам нужны только определенные поля.

```
cut [-options][path]
```

В нашем образце файла мы имеем наши данные в 3 столбцах, первый-имя, второй-плод и третий-количество. Скажем, нам нужна была только первая колонка.

```
-d, --delimiter=DELIM
 use DELIM instead of TAB for field delimiter
-f, --fields=LIST
 select only these fields; also print any line that contains no
 delimiter character, unless the -s option is specified
```

cut по умолчанию используется символ табуляции в качестве разделителя (разделитель, -d) для идентификации полей (-f). В нашем файле мы использовали один пробел, поэтому нам нужно сказать cut, чтобы использовать его вместо этого.

```
azat@pop-os:~/Temp$ cut -f 1 -d ' ' mysampleddata.txt
Fred
Susy
Mark
```

```
Robert
Terry
Lisa
Susy
Mark
Anne
Greg
Oliver
Betty
azat@pop-os:~/Temp$
```

Если мы хотим 2 или более полей, то мы разделяем их запятой, как показано ниже.

```
azat@pop-os:~/Temp$ cut -f 1,2 -d ' ' mysampleddata.txt
Fred apples
Susy oranges
Mark watermellons
Robert pears
Terry oranges
Lisa peaches
Susy oranges
Mark grapes
Anne mangoes
Greg pineapples
Oliver rockmellons
Betty limes
azat@pop-os:~/Temp$
```

## sed

sed расшифровывается как **Stream Editor**, и это эффективно позволяет нам выполнять поиск и замену на наших данных. Это довольно мощная команда, но мы будем использовать ее здесь в базовом формате.

```
sed <expression> [path]
```

Базовое выражение имеет следующий формат:

```
s/search/replace/g
```

Инициалы **s** означают замену и указывают действие для выполнения (есть и другие, но пока мы будем держать его простым). Затем между первой и второй косыми чертами ( / ) помещаем то, что мы помещаем то, что **ищем** ( **searching** ) Затем между второй и третьей косыми чертами-то, чем мы хотим его **заменить** ( **replace** ). **g** в конце означает **глобальный** и является необязательным.

```
azat@pop-os:~/Temp$ sed 's/oranges/bananas/g' mysampleddata.txt
```

```
Fred apples 20
Susy bananas 5
Mark watermellons 12
Robert pears 4
Terry bananas 9
Lisa peaches 7
Susy bananas 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
azat@pop-os:~/Temp$
```

Важно отметить, что **sed** идентифицирует не слова, а строки символов.

## uniq

uniq расшифровывается как unique, и его задача-удалить дублирующиеся строки из данных. Однако одно ограничение состоит в том, что эти линии должны быть смежными (т. е.)

```
uniq [options] [path]
```

```
azat@pop-os:~/Temp$ pwd
/home/azat/Temp
azat@pop-os:~/Temp$ ls
azat@pop-os:~/Temp$ cat mysampleddata.txt
Fred apples 20
Susy oranges 5
Susy oranges 5
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
azat@pop-os:~/Temp$ uniq mysampleddata.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert pears 4
```



```
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
azat@pop-os:~/Temp$
```

## tac

Ребята из Linux известны своим забавным чувством юмора. Программа tac на самом деле cat в обратном порядке. Он был назван так, как он делает противоположность кошки. Учитывая данные он будет печатать последнюю строку первым, вплоть до первой линии.

```
tac [path]
```

```
azat@pop-os:~/Temp$ tac mysampleddata.txt
Betty limes 14
Oliver rockmellons 2
Greg pineapples 3
Anne mangoes 7
Mark grapes 39
Mark grapes 39
Susy oranges 12
Lisa peaches 7
Terry oranges 9
Robert pears 4
Mark watermellons 12
Susy oranges 5
Susy oranges 5
Susy oranges 5
Fred apples 20
azat@pop-os:~/Temp$
```

## Others

Есть много и намного больше команд linux для изучения, вы можете искать их и изучать, это было бы действительно круто ~

- awk
- diff
- .....

## Краткое изложенике

command	action
<code>head</code>	View the first n lines of the data.
<code>tail</code>	View the last n lines of the data.
<code>sort</code>	Organises the data into order.
<code>nl</code>	Print line numbers before the data.
<code>wc</code>	Print a count of lines, words, and characters.
<code>cut</code>	Cut the data into fields and only display the specified fields.
<code>sed</code>	Do a search and replace on the data.
<code>uniq</code>	Remove duplicated lines.
<code>tac</code>	Print the data in reverse order.

## Задания

Давайте скажем некоторые данные.

- Во-первых, вы можете сделать файл с данными, похожими на наш образец файла.
- Теперь поиграйте с каждой из программ, которые мы рассмотрели выше. Убедитесь, что вы используете оба относительных и абсолютных пути.
- Посмотрите на справочную страницу для каждой из программ и попробуйте хотя бы 2 команды командной строки варианты для них.

## Гrep и регулярные выражения!

### Регулярное выражение

В этой части урока мы рассмотрим другой фильтр, который является довольно мощным в сочетании с концепцией, которая называется регулярными выражениями или регулярное выражение для краткости. Регулярные выражения похожи на подстановочные знаки, которые мы изучали в предыдущем разделе. Они позволяют нам создать шаблон. Однако они немного сильнее. Регулярные выражения обычно используются для идентификации и обработки определенных фрагментов данных. Например, мы можем пожелать идентифицировать каждую строку, содержащую адрес электронной почты или url в наборе данных.

### egrep

Egrep - это программа, которая будет искать заданный набор данных и печатать каждую строку, содержащую заданный шаблон, это расширение программы, называемой grep. Есть много вариантов egrep, например, опция -v говорит grep вместо этого печатать каждую строку, которая не соответствует шаблону.

```
egrep [command line options] <pattern> [path]
```

Допустим, мы хотели бы идентифицировать каждую строку, содержащую строку **mellon**

```
egrep 'mellon' mysampleddata.txt
```

```
azat@pop-os:~/Temp$ cat mysampleddata.txt
Fred apples 20
Susy oranges 5
Susy oranges 5
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
azat@pop-os:~/Temp$ egrep 'mellon' mysampleddata.txt
Mark watermellons 12
Oliver rockmellons 2
azat@pop-os:~/Temp$
```

Основное поведение egrep заключается в том, что он будет печатать всю строку для каждой строки, содержащей строку символов, соответствующих заданному шаблону. Важно отметить, что мы ищем не слово, а строку символов.

Иногда мы хотим знать не только, какие линии совпадают, но и их номер.

```
azat@pop-os:~/Temp$ egrep -n 'mellon' mysampleddata.txt
5:Mark watermellons 12
14:Oliver rockmellons 2
```

Или, может быть, мы не заинтересованы в том, чтобы видеть совпадающие линии, но хотим знать, сколько линий совпало.

```
azat@pop-os:~/Temp$ egrep -c 'mellon' mysampleddata.txt
2
```

## Обзор регулярных выражений

Здесь мы не будем много говорить о регулярных выражениях, а только некоторые основы и обзор. Вы должны иметь возможность искать регулярные выражения в google и подробно изучать другие ресурсы.

Вы также можете найти здесь очень полезные электронные книги [AzatAI CS BOOKS](#)

Вот основные строительные блоки регулярных выражений ниже, а затем следуйте с набором примеров, чтобы продемонстрировать их использование

- **.** (**dot**) - один символ.
- **?** - следующий символ соответствует только 0 или 1 раз.
- **\*** - следующий символ соответствует 0 или более раз.
- **+** - соответствует 1 или более раз
- **{n}** - соответствует ровно n раз.
- **{n,m}** - соответствует не менее n раз и не более m раз.
- **[agd]** соответствует одному из тех, что заключены в квадратные скобки.
- **[^agd]** этот символ не входит в число тех, что заключены в квадратные скобки.
- **[c-f]** - тире в квадратных скобках работает как диапазон. От c до f, тогда это c, d, e или f.
- **()** - сгруппируйте несколько персонажей, чтобы вести себя как один.
- **|** (**pipe symbol**) - оригинал или операция.
- **^** - соответствует началу строки.
- **\$** - соответствует концу строки.

## Примеры

Допустим, мы хотим идентифицировать любую строку с двумя или более гласными подряд.

```
azat@pop-os:~/Temp$ egrep '[aeiou]{2,}' mysampleddata.txt
Robert pears 4
Lisa peaches 7
Anne mangoes 7
Greg pineapples 3
```

Как насчет любой строки с 2 на ней, которая не является концом строки

```
azat@pop-os:~/Temp$ egrep '2.+' mysampleddata.txt
Fred apples 20
```

Число 2 как последний символ в строке

```
azat@pop-os:~/Temp$ egrep '2.+' mysampleddata.txt
Fred apples 20
azat@pop-os:~/Temp$ egrep '2$' mysampleddata.txt
Mark watermellons 12
Susy oranges 12
Oliver rockmellons 2
```

А теперь каждая строка, содержащая либо 'is' or 'go' or 'or'.

```
azat@pop-os:~/Temp$ egrep '2.+' mysampledата.txt
Fred apples 20
azat@pop-os:~/Temp$ egrep '2$' mysampledата.txt
Mark watermellons 12
Susy oranges 12
Oliver rockmellons 2
azat@pop-os:~/Temp$ egrep 'is|or|go' mysampledата.txt
Susy oranges 5
Susy oranges 5
Susy oranges 5
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Anne mangoes 7
```

Может быть, мы хотим видеть заказы для всех, чье имя начинается с A-K.

```
azat@pop-os:~/Temp$ egrep '2.+' mysampledата.txt
Fred apples 20
azat@pop-os:~/Temp$ egrep '2$' mysampledата.txt
Mark watermellons 12
Susy oranges 12
Oliver rockmellons 2
azat@pop-os:~/Temp$ egrep 'is|or|go' mysampledата.txt
Susy oranges 5
Susy oranges 5
Susy oranges 5
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Anne mangoes 7
azat@pop-os:~/Temp$ egrep '^[A-K]' mysampledата.txt
Fred apples 20
Anne mangoes 7
Greg pineapples 3
Betty limes 14
```

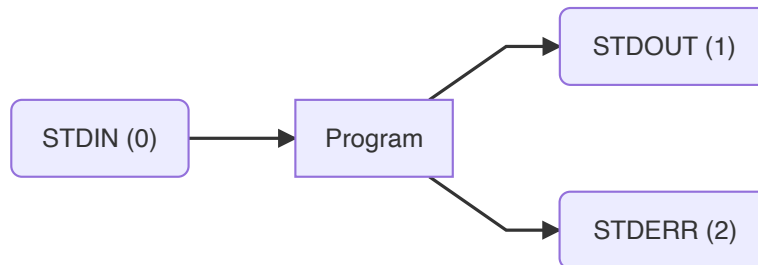
## Конвейеризация и перенаправление

---

### Вступление

Каждая команда, которую мы запускаем в командной строке, автоматически имеет три потока данных, подключенных к ней.

- STDIN (0) - Стандартный ввод (**Standard input**)(данные, подаваемые в программу).
- STDOUT (1) - Стандартный вывод (**Standard output**)(данные, напечатанные программой, по умолчанию на терминал)
- STDERR (2) - Стандартная ошибка (**Standard error**)(для сообщений об ошибках также по умолчанию используется терминал)



Конвейеризация и перенаправление - это средства, с помощью которых мы можем соединять эти потоки между программами и файлами, чтобы направлять данные интересными и полезными способами.

Ниже мы продемонстрируем конвейеризацию и перенаправление на нескольких примерах, но эти механизмы будут работать с каждой программой в командной строке, а не только с теми, которые мы использовали в примерах.

## Перенаправление в файл

Обычно мы получаем наши выходные данные на экране, что удобно большую часть времени, но иногда мы можем сохранить их в файл, чтобы сохранить как запись, передать в другую систему или отправить кому-то еще. Оператор больше, чем (>) указывает командной строке, что мы хотим, чтобы вывод программы (или все, что она отправляет в STDOUT) был сохранен в файле, а не напечатан на экране. Давайте рассмотрим пример.

```
azat@pop-os:~/Temp$ ls
Almaty blahblah.txt file2.txt foofoo main.azt ok.request
azt.md bob.md firstfile images mysampled.txt secret.mp4
blafiii.txt demo fool love.mp3 ok video.mpeg
azat@pop-os:~/Temp$ ls > myoutput
azat@pop-os:~/Temp$ cat myoutput
Almaty
azt.md
blafiii.txt
blahblah.txt
bob.md
demo
file2.txt
firstfile
```

```
fool
foofoo
images
love.mp3
main.azt
myoutput
mysampleddata.txt
ok
ok.request
secret.mp4
video.mepg
azat@pop-os:~/Temp$
```

## Сохранение в существующий файл

Если мы перенаправим на несуществующий файл, он будет создан автоматически для нас. Если мы сохраняем в файл, который уже существует, однако, то его содержимое будет очищено, а затем новый вывод сохранен в нем.

Вместо этого мы можем добавить новые данные в файл с помощью оператора double greater than (>>).

```
azat@pop-os:~/Temp$ file azt.md >>myoutput
azat@pop-os:~/Temp$ tac myoutput
azt.md: ASCII text, with very long lines
video.mepg
secret.mp4
ok.request
ok
mysampleddata.txt
myoutput
main.azt
love.mp3
images
foofoo
fool
firstfile
file2.txt
demo
bob.md
blahblah.txt
blafiii.txt
azt.md
Almaty
azat@pop-os:~/Temp$
```

## Перенаправление из файла

Если мы используем оператор less than (<), то мы можем отправить данные другим способом. Мы будем считывать данные из файла и передавать их в программу через его поток STDIN.

```
azat@pop-os:~/Temp$ wc -l myoutput
20 myoutput
azat@pop-os:~/Temp$ wc -l < myoutput
20
```

Многие программы (как мы видели в предыдущих разделах) позволяют нам предоставить файл в качестве аргумента командной строки, и он будет читать и обрабатывать содержимое этого файла.

Всякий раз, когда мы используем перенаправление или конвейер, данные отправляются анонимно. Таким образом, в приведенном выше примере wc получил некоторый контент для обработки, но он не знает, откуда он взялся, поэтому он не может печатать эту информацию. В результате этот механизм часто используется для того, чтобы получить дополнительные данные (которые могут и не потребоваться), которые не будут напечатаны.

Мы можем легко объединить две формы перенаправления, которые мы видели до сих пор, в одну команду, как показано в примере ниже.

```
azat@pop-os:~/Temp$ wc -l myoutput
20 myoutput
azat@pop-os:~/Temp$ wc -l < myoutput
20
azat@pop-os:~/Temp$ wc -l < azt.md >>myoutput
azat@pop-os:~/Temp$ wc -l tac myoutput
```

```
azat@pop-os:~/Temp$ wc -l < azt.md >>myoutput
azat@pop-os:~/Temp$ tac myoutput
39
azt.md: ASCII text, with very long lines
video.mepg
secret.mp4
ok.request
ok
mysampleddata.txt
myoutput
main.azt
love.mp3
images
foofoo
fool
firstfile
```



```
file2.txt
demo
bob.md
blahblah.txt
blafiii.txt
azt.md
Almaty
```

## Перенаправить поток STDERR

Теперь давайте посмотрим на третий поток, который является стандартной ошибкой или STDERR. Три потока на самом деле имеют номера, связанные с ними (в скобках в списке вверху страницы). STDERR-это поток номер 2, и мы можем использовать эти числа для идентификации потоков. Если мы поместим число перед оператором `>`, то он перенаправит этот поток (если мы не используем число, как мы делали до сих пор, то по умолчанию используется поток 1).

```
azat@pop-os:~/Temp$ cd uthjg 2>errors.txt
azat@pop-os:~/Temp$ cat errors.txt
bash: cd: uthjg: No such file or directory
```

Возможно, мы хотим сохранить как обычный вывод, так и сообщения об ошибках в одном файле. Это можно сделать, перенаправив поток STDERR в поток STDOUT и перенаправив STDOUT в файл.

Сначала мы перенаправляем в файл, а затем перенаправляем поток ошибок. Мы идентифицируем перенаправление на поток, помещая `&` перед номером потока (в противном случае он перенаправил бы на файл с именем 1).

```
azat@pop-os:~/Temp$ ls -l ddaisd > myoutput2 2>&1
azat@pop-os:~/Temp$ cat myoutput2
ls: cannot access 'ddaisd': No such file or directory
```

## Конвейеризация

До сих пор мы имели дело с отправкой данных в файлы и из файлов. Теперь рассмотрим механизм передачи данных из одной программы в другую. Он называется `pipng`, и оператор, который мы используем, - это `( | )` (находится над обратной косой чертой `( \ )` на большинстве клавиатур).

Что делает этот оператор, так это передает выходные данные из программы слева в качестве входных данных в программу справа.

```
azat@pop-os:~/Temp$ ls
Almaty bob.md firstfile love.mp3 mysampledata.txt video.mpeg
azt.md demo fool main.azt ok
blafiii.txt errors.txt foofoo myoutput ok.request
blahblah.txt file2.txt images myoutput2 secret.mp4
azat@pop-os:~/Temp$ ls | head -3
Almaty
azt.md
blafiii.txt
```

Мы можем передавать вместе столько программ, сколько захотим.

```
azat@pop-os:~/Temp$ ls
Almaty bob.md firstfile love.mp3 mysampledata.txt video.mpeg
azt.md demo fool main.azt ok
blafiii.txt errors.txt foofoo myoutput ok.request
blahblah.txt file2.txt images myoutput2 secret.mp4
azat@pop-os:~/Temp$ ls | head -3
Almaty
azt.md
blafiii.txt
azat@pop-os:~/Temp$ ls | head -3 | tail -1
blafiii.txt
```

Любые аргументы командной строки, которые мы предоставляем для программы, должны быть рядом с этой программой.

Вы также можете комбинировать трубы и перенаправление.

```
azat@pop-os:~$ ls | head -3 | tail -1 >myoutput
azat@pop-os:~$ cat myoutput
Downloads
```

Есть много вещей, которые вы можете достичь с помощью трубопроводов, и это лишь некоторые из них. С опытом и немного творческого мышления я уверен, что вы найдете гораздо больше способов использовать трубопроводы, чтобы сделать вашу жизнь проще.

В этом примере мы сортируем список каталогов так, чтобы все каталоги были перечислены первыми.

```
azat@pop-os:~/Temp$ ls -l /etc |tail -n +2 | sort
```

`-n, --lines=[+]NUM`

output the last NUM lines, instead of the last 10; or use `-n +NUM` to output starting with line NUM

```
azat@pop-os:~/Temp$ ls -l
azat@pop-os:~/Temp$ ls -l | tail
azat@pop-os:~/Temp$ ls -l | tail -n +2 | sort
```

**ls -l** длинный список каталогов. Как всегда, первая строка-это какая-то информация вроде `total 36`.

**tail** является противоположностью **head**, он перечисляет все от начала до конца. И `-n +2` значит, берем данные со второй строки, а по которой не хотим `total 36`.

**sort** сначала сортирует результаты по каталогам.

Как насчет того, если мы используем `head` скорее, чем `tail` ?

```
azat@pop-os:~/Temp$ ls -l | head | sort
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 blafiii.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 blahblah.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 bob.md
-rw-r--r-- 1 azat azat 0 Jan 9 20:16 Almaty
-rw-r--r-- 1 azat azat 0 Jan 9 20:21 file2.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:21 fool
-rw-r--r-- 1 azat azat 3399 Jan 9 12:17 azt.md
-rw-r--r-- 1 azat azat 43 Jan 10 18:39 errors.txt
-rw-r--r-- 1 azat azat 64 Jan 9 11:44 firstfile
total 36
```

`head` дозы не имеют `-n +NUM` вариант.

В этом примере мы будем меньше подавать выходные данные программы в программу, чтобы нам было легче ее просматривать.

```
azat@pop-os:~/Temp$ ls -l | less
```

```
ls -l | less

total 36
-rw-r--r-- 1 azat azat 0 Jan 9 20:16 Almaty
-rw-r--r-- 1 azat azat 3399 Jan 9 12:17 azt.md
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 blafiii.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 blahblah.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 bob.md
-rw-r--r-- 1 azat azat 43 Jan 10 18:39 errors.txt
-rw-r--r-- 1 azat azat 0 Jan 9 20:21 file2.txt
-rw-r--r-- 1 azat azat 64 Jan 9 11:44 firstfile
-rw-r--r-- 1 azat azat 0 Jan 9 20:21 fool
```

```
drwxr-xr-x 2 azat azat 4096 Jan 9 10:16 foofoo
drwxr-xr-x 2 azat azat 4096 Jan 9 20:41 images
-rw-r--r-- 1 azat azat 0 Jan 9 20:18 love.mp3
-rwxr--r-- 1 azat azat 0 Jan 9 21:15 main.azt
-rw-r--r-- 1 azat azat 214 Jan 10 18:36 myoutput
-rw-r--r-- 1 azat azat 54 Jan 10 18:42 myoutput2
-rw-r--r-- 1 azat azat 242 Jan 9 22:35 mysampleddata.txt
drwxr-xr-x 3 azat azat 4096 Jan 9 10:19 ok
-rw-r--r-- 1 azat azat 0 Jan 9 20:10 ok.request
-rw-r--r-- 1 azat azat 0 Jan 9 20:18 secret.mp4
-rw-r--r-- 1 azat azat 0 Jan 9 20:20 video.mepg
(END)
```

Определите все файлы в вашем домашнем каталоге, для которых группа имеет разрешение на запись.

```
ls -l ~ |grep '^.....w'
```

## Краткое содержание

- > сохраните выходные данные в файл.
- >> Добавьте выходные данные в файл.
- < Чтение входных данных из файла.
- 2> перенаправить сообщение об ошибке.
- | Отправить выходные данные из одной программы в качестве входных данных в другую программу.
- Потоки каждая программа, которую вы можете запустить в командной строке, имеет 3 потока: STIN, STDOUT и STDERR.

## Задание

Давайте искажим некоторые данные:

- Прежде всего, поэкспериментируйте с сохранением выходных данных из различных команд в файл. Перепишите файл и добавьте к нему также. Убедитесь, что вы используете как абсолютный, так и относительный пути, как вы идете.
- Теперь посмотрите, можете ли вы перечислить только 20-й последний файл в каталоге / etc.
- Наконец, посмотрите, можете ли вы подсчитать, сколько файлов и каталогов у вас есть разрешение на выполнение в вашем домашнем каталоге.

## Управление процессами

### Вступление

Программа - это серия инструкций, которые говорят компьютеру, что делать. Когда мы запускаем программу, эти инструкции копируются в память, а пространство выделяется для переменных и других вещей, необходимых для управления ее выполнением. Это запущенный экземпляр программы называется процессом и это процессы которыми мы управляем

## top Что работает в данный момент?

Linux, как и многие другие операционные системы, является многозадачной операционной системой. Это означает, что многие процессы выполняются одновременно, и даже многие пользователи работают одновременно. Если мы хотим получить то, что в данный момент происходит в системе, мы можем использовать команду `top`.

```
azat@pop-os:~/Temp$ ls -l ~ | grep '^.....W'
azat@pop-os:~/Temp$ top
```

```
top - 11:20:31 up 1:25, 1 user, load average: 0.13, 0.03, 0.01
Tasks: 203 total, 1 running, 202 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.6 us, 3.1 sy, 0.0 ni, 93.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1981.1 total, 143.2 free, 836.4 used, 1001.6 buff/cache
MiB Swap: 4095.5 total, 4095.5 free, 0.0 used. 975.5 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1289	azat	20	0	159400	45624	12152	S	4.7	2.2	0:29.70	Xorg
1695	azat	-2	0	2692344	333136	73660	S	4.7	16.4	0:33.59	gnome-she+
2568	azat	20	0	631112	47368	33792	S	2.3	2.3	0:05.24	gnome-ter+
744	message+	20	0	8800	5860	3856	S	0.7	0.3	0:14.97	dbus-daem+
1959	azat	20	0	550708	28420	19560	S	0.7	1.4	0:21.32	prlcc
1	root	20	0	101580	10980	8012	S	0.3	0.5	0:01.09	systemd
1974	azat	20	0	350160	14804	9812	S	0.3	0.7	0:02.44	prlsga
18198	azat	20	0	21560	3752	3112	R	0.3	0.2	0:00.04	top
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp

```

 4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00
rcu_par_gp
 6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00
kworker/0+
 9 root 0 -20 0 0 0 I 0.0 0.0 0:00.00
mm_percpu+
 10 root 20 0 0 0 0 S 0.0 0.0 0:00.05
ksoftirqd+
 11 root 20 0 0 0 0 I 0.0 0.0 0:02.15 rcu_sched

 12 root rt 0 0 0 0 S 0.0 0.0 0:00.02
migration+
 13 root -51 0 0 0 0 S 0.0 0.0 0:00.00
idle_inje+

```

Некоторые задачи (процесс) - это спящие средства, они ждут, пока не произойдет определенное событие, на которое они затем будут действовать.

Строка 3 - использование процессора

Строка 4 - это использование памяти. (Вы должны беспокоиться об этом)

Строка 5 - это виртуальная память: (SWAP), вы должны увеличить ее размер, если ваш компьютер использовал слишком много из них.

Как насчет поиска в Google для SWAP?

Строка 7 - список наиболее ресурсоемких процессов в системе (в порядке использования ресурсов). Этот список будет обновляться в режиме реального времени и поэтому интересно наблюдать, чтобы получить представление о том, что происходит в вашей системе.

Два важных столбца для рассмотрения-это использование памяти и процессора. Если любой из них высок для конкретного процесса в течение определенного периода времени, возможно, стоит посмотреть, почему это так. Столбец пользователь показывает, кому принадлежит процесс, а столбец PID определяет идентификатор процесса.

“top” дает вам представление о системе в реальном времени и показывает только количество процессов, расположенных на экране.

## ps процесс

Другая программа для рассмотрения процесса называется `ps`, что означает процесс. В его обычном использовании он покажет вам только процессы, запущенные в вашем текущем терминале (что обычно не очень много). Если мы добавим аргумент `aux`, то он покажет полное представление, которое немного более полезно

```

azat@pop-os:~/Temp$ ps
 PID TTY TIME CMD
 2580 pts/0 00:00:00 bash
 17963 pts/0 00:00:00 ps

```

```

azat@pop-os:~/Temp$ ps a
 PID TTY START TIME COMMAND
 1284 tty2 Ssl+ 0:00 /usr/lib/gbm3/gbm-x-session --run-script env GNOME_SH
 1289 tty2 Rl+ 1:14 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1
 1435 tty2 Sl+ 0:00 /usr/lib/gnome-session/gnome-session-binary --systemd
 2580 pts/0 Ss 0:00 bash
 4784 pts/1 Ss 0:00 bash
17399 pts/1 S+ 0:00 man ps
17412 pts/1 S+ 0:00 pager
17973 pts/0 R+ 0:00 ps a
azat@pop-os:~/Temp$ ps au
USER PID %CPU %MEM VSZ RRS TTY STAT START TIME COMMAND
azat 1284 0.0 0.3 173640 6420 tty2 Ssl+ 10:13 0:00
/usr/lib/gdm3/g
azat 1289 0.4 2.2 159400 45624 tty2 Sl+ 10:13 1:14
/usr/lib/xorg/X
azat 1435 0.0 0.7 201808 15380 tty2 Sl+ 10:13 0:00
/usr/lib/gnome-
azat 2580 0.0 0.2 20648 5324 pts/0 Ss 10:14 0:00 bash
azat 4784 0.0 0.2 20644 5040 pts/1 Ss 10:22 0:00 bash
azat 17399 0.0 0.1 19364 3864 pts/1 S+ 14:46 0:00 man ps
azat 17412 0.0 0.1 18188 2548 pts/1 S+ 14:46 0:00 pager
azat 17983 0.0 0.1 21216 3284 pts/0 R+ 14:50 0:00 ps au
azat@pop-os:~/Temp$ ps aux
azat@pop-os:~/Temp$

```

```

azat@pop-os:~/Temp$ ps
 PID TTY TIME CMD
 2580 pts/0 00:00:00 bash
18117 pts/0 00:00:00 ps

```

`ps` выводит текущий терминальный процесс.

```

azat@pop-os:~/Temp$ ps a
 PID TTY STAT TIME COMMAND
 1284 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SH
 1289 tty2 Sl+ 1:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1
 1435 tty2 Sl+ 0:00 /usr/lib/gnome-session/gnome-session-binary --
systemd
 2580 pts/0 Ss 0:00 bash
 4784 pts/1 Ss 0:00 bash
17399 pts/1 S+ 0:00 man ps
17412 pts/1 S+ 0:00 pager
18247 pts/0 R+ 0:00 ps a

```

`ps a` показывает процесс для всех пользователей.(Процесс прикреплен к терминалу).

```
azat@pop-os:~/Temp$ ps au
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
azat 1284 0.0 0.3 173640 6420 tty2 Ssl+ 10:13 0:00 /usr/lib/gdm3/g
azat 1289 0.4 2.2 159912 45624 tty2 Sl+ 10:13 1:16 /usr/lib/xorg/X
azat 1435 0.0 0.7 201808 15380 tty2 Sl+ 10:13 0:00 /usr/lib/gnome-
azat 2580 0.0 0.2 20648 5324 pts/0 Ss 10:14 0:00 bash
azat 4784 0.0 0.2 20644 5040 pts/1 Ss 10:22 0:00 bash
azat 17399 0.0 0.1 19364 3864 pts/1 S+ 14:46 0:00 man ps
azat 17412 0.0 0.1 18188 2548 pts/1 S+ 14:46 0:00 pager
azat 18447 0.0 0.1 21216 3396 pts/0 R+ 14:52 0:00 ps au
azat@pop-os:~/Temp$
```

`ps au` показать процесс для всех пользователей, а также показать пользователя/владельца процесса.

```
azat@pop-os:~/Temp$ ps aux
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.5 101580 10980 ? Ss 10:13 0:01 /sbin/init
spld
root 2 0.0 0.0 0 0 ? S 10:13 0:00 [kthreadd]
root 3 0.0 0.0 0 0 ? I< 10:13 0:00 [rcu_gp]
root 4 0.0 0.0 0 0 ? I< 10:13 0:00 [rcu_par_gp]
root 6 0.0 0.0 0 0 ? I< 10:13 0:00
[kworker/0:0H-k
root 9 0.0 0.0 0 0 ? I< 10:13 0:00
[mm_percpu_wq]
root 10 0.0 0.0 0 0 ? S 10:13 0:00 [ksoftirqd/0]
root 11 0.0 0.0 0 0 ? R 10:13 0:07 [rcu_sched]
root 12 0.0 0.0 0 0 ? S 10:13 0:00 [migration/0]
root 13 0.0 0.0 0 0 ? S 10:13 0:00
[idle_inject/0]
root 14 0.0 0.0 0 0 ? S 10:13 0:00 [cpuhp/0]
root 15 0.0 0.0 0 0 ? S 10:13 0:00 [cpuhp/1]
root 16 0.0 0.0 0 0 ? S 10:13 0:00
[idle_inject/1]
root 17 0.0 0.0 0 0 ? S 10:13 0:00 [migration/1]
root 18 0.0 0.0 0 0 ? S 10:13 0:00 [ksoftirqd/1]
root 20 0.0 0.0 0 0 ? I< 10:13 0:00
[kworker/1:0H-k
root 21 0.0 0.0 0 0 ? S 10:13 0:00 [kdevtmpfs]
```



`ps aux` показать процесс для всех пользователей, включая владельца процесса, а также показать процессы, которые не подключены к терминалу  
Это дает довольно много выходных данных, поэтому люди обычно передают выходные данные в `grep`, чтобы отфильтровать только те данные, которые они ищут. Мы увидим в следующем бите пример этого.

## **kill** Убийство разбившегося процесса

Когда программа выходит из строя, это может быть довольно раздражающим. Тем не менее, мы можем легко “убить” процесс, а затем снова открыть его, например, предположим, что ваш Firefox застрял, когда мы пытаемся открыть <https://azat.ai>, и поэтому для начала нам нужно определить идентификатор процесса Firefox:

```
azat@pop-os:~/Temp$ ps aux | grep 'firefox'
azat@pop-os:~/Temp$
```

```
azat@pop-os:~/Temp$ ps aux | grep 'firefox'
azat 19751 3.2 13.5 2775580 274256 pts/1 Sl+ 14:58 0:04
/usr/lib/firefox/firefox
azat 19795 1.0 7.8 2482740 158632 pts/1 Sl+ 14:59 0:01
/usr/lib/firefox/firefox -contentproc -childID 1 -isForBrowser -prefsLen 1 -
prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 19832 0.4 5.2 2397172 106896 pts/1 Sl+ 14:59 0:00
/usr/lib/firefox/firefox -contentproc -childID 2 -isForBrowser -prefsLen 6975
-prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 19937 0.2 4.1 2384400 83704 pts/1 Sl+ 14:59 0:00
/usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 11762
-prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 20290 0.0 0.0 18664 980 pts/0 S+ 15:01 0:00 grep --
color=auto firefox
```

Как мы легко можем понять, основной идентификатор процесса Firefox- `19571` , давайте уьем его сейчас (сначала вежливо попросим программу выйти из нее) :

```
azat@pop-os:~/Temp$ ps aux|grep 'firefox'
azat@pop-os:~/Temp$ kill 19751
azat@pop-os:~/Temp$
```

```

azat@pop-os:~/Temp$ ps aux|grep 'firefox'
azat 19751 1.6 13.8 2775580 280788 pts/1 Sl+ 14:58 0:04
/usr/lib/firefox/firefox
azat 19795 0.4 7.8 2482740 158972 pts/1 Sl+ 14:59 0:01
/usr/lib/firefox/firefox -contentproc -childID 1 -isForBrowser -prefsLen 1 -
prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 19832 0.2 5.2 2397172 107024 pts/1 Sl+ 14:59 0:00
/usr/lib/firefox/firefox -contentproc -childID 2 -isForBrowser -prefsLen 6975
-prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 19937 0.1 4.1 2384400 83704 pts/1 Sl+ 14:59 0:00
/usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 11762
-prefMapSize 202500 -parentBuildID 20191205203726 -greomni
/usr/lib/firefox/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appdir
/usr/lib/firefox/browser 19751 true tab
azat 20738 0.0 0.0 18664 848 pts/0 S+ 15:03 0:00 grep --
color=auto firefox
azat@pop-os:~/Temp$ kill 19751

```

```
kill [signal] <PID>
```

Это число рядом с владельцем процесса, которое является PID (Process ID). Мы будем использовать это, чтобы определить, какой процесс убить. Для этого мы используем программу, которая называется **kill**.

Ладно, как ты думаешь? Конечно не все программы достаточно вежливы, что некоторые из них будут не совсем, давайте проверим, работает ли Firefox до сих пор

```

azat@pop-os:~/Temp$ ps aux|grep 'firefox'
azat 21400 0.0 0.0 18664 984 pts/0 S+ 15:07 0:00 grep --color=auto firefox
azat@pop-os:~/Temp$

```

Вы знаете, что это довольно плохо, что я попросил Firefox довольно вежливо, но, однако, это не полностью, так что на этот раз, я хочу убедиться, что Firefox убит, полностью.

Когда мы просто используем `kill <PID>`, на самом деле kill посылает процессу сигнал по умолчанию `1`, очень вежливо прося его выйти из процесса. Если это не сработает, мы можем немного не вежливо вообще, например, изменить сигнал 9, что на самом деле означает, что процесс хорошо и действительно прошел.

Для вашего понимания, вот таблица сигнальных действий Linux и их переносимых номеров.

Signal	Portable number	Default Action	Description
SIGHUP	1	Terminate	Hangup
SIGINT	2	Terminate	Terminal interrupt signal
SIGQUIT	3	Terminate (core dump)	Terminal quit signal
SIGILL	4	Terminate (core dump)	Illegal instruction
SIGTRAP	5	Terminate (core dump)	Trace/breakpoint trap
SIGABRT	6	Terminate (core dump)	Process abort signal
SIGBUS	7	Terminate (core dump)	Bus error (bad memory access)
SIGFPE	8	Terminate (core dump)	Erroneous arithmetic operation
SIGKILL	9	Terminate	Kill (cannot be caught or ignored)
..	..	..	..

- Control+C (control character `intr`) посылает SIGINT, который прерывает приложение. Обычно это приводит к прерыванию, но это зависит от приложения, чтобы решить
- Control+Z (control character `susp`) отправляет SIGTSTP в приложение переднего плана, эффективно помещая его в фоновый режим, приостановленный. Это полезно, если вам нужно вырваться из чего-то вроде редактора, чтобы пойти и захватить некоторые данные, которые вам нужны. Вы можете вернуться в приложение, запустив `fg` (или `%x` где `x` номер задания, как показано в разделе `jobs`).

Обычные пользователи могут убивать только те процессы, владельцем которых они являются. Пользователь root в системе может убить любые процессы.

В Linux на самом деле работает несколько виртуальных консолей. Большую часть времени мы видим только консоль 7, которая является графическим интерфейсом, но мы можем легко добраться до других. Для переключения между консолями используется последовательность клавиш **CTRL + ALT + F**.

## Задания переднего плана и фона

Вероятно, вам не нужно будет делать слишком много задания с передним планом и фоном, но стоит знать о них только для редких случаев. Когда мы запускаем программу, как правило, они выполняются на переднем плане. Большинство из них также завершаются за долю секунды. Может быть, мы хотим начать процесс, который займет немного времени и с радостью сделает это без нашего вмешательства (например, обработка очень большого текстового файла или компиляция программы). Что мы можем сделать это запустить программу в фоновом режиме, а затем мы можем продолжить работу. Мы продемонстрируем это с помощью программы под названием `sleep`. Вся доза сна - это подождать заданное количество секунд, а затем бросить курить. Мы также можем использовать программу под названием `jobs`, которая перечисляет текущие фоновые задания для нас

```
jobs
```

```
azat@pop-os:~/Temp$ sleep 5
azat@pop-os:~/Temp$ jobs
azat@pop-os:~/Temp$

azat@pop-os:~/Temp$ ps auxlgrep 'firefox'
azat 21400 0.0 0.0 18664 984 pts/0 S+ 15:07 0:00 grep --color=auto firefox
azat@pop-os:~/Temp$ clear
```

```
azat@pop-os:~/Temp$ sleep 5
azat@pop-os:~/Temp$ jobs
```

Теперь мы выполним ту же команду, что и выше, но вместо этого поставим амперсанд ( & ) в конце команды, после чего мы обращаемся терминалу запустить этот процесс в фоновом режиме.

```
azat@pop-os:~/Temp$ sleep 5 &
[1] 27824
azat@pop-os:~/Temp$
[1]+ Done sleep 5
azat@pop-os:~/Temp$
```

```
azat@pop-os:~/Temp$ sleep 5 &
[1] 27824
azat@pop-os:~/Temp$
[1]+ Done sleep 5
azat@pop-os:~/Temp$
```

Мы также можем перемещать задания между передним и задним планом. Если вы нажмете **CTRL+Z**, то текущий процесс переднего плана будет приостановлен и перемещен в фоновый режим. Затем мы можем использовать программу `ad` под названием `fg`, которая расшифровывается как `foreground`, чтобы вывести фоновый процесс на передний план.

```
fg <job number>
```

```
azat@pop-os:~/Temp$ sleep 15 &
[1] 30089
azat@pop-os:~/Temp$ sleep 10
^Z
[2]+ Stopped sleep 10
azat@pop-os:~/Temp$ jobs
[1]- Running sleep 15 &
[2]+ Stopped sleep 10
azat@pop-os:~/Temp$ fg 2
sleep 10
[1] Done sleep 15
azat@pop-os:~/Temp$
```

```
azat@pop-os:~/Temp$ sleep 15 &
[1] 30089
azat@pop-os:~/Temp$ sleep 10
^Z
[2]+ Stopped
azat@pop-os:~/Temp$ jobs
[1]- Running
[2]+ Stopped
azat@pop-os:~/Temp$ fg 2
sleep 10
[1] Done
azat@pop-os:~/Temp$
```

## Краткое содержание

- **top**  
Просмотр в режиме реального времени данных о процессах, запущенных в системе
- **ps**  
Получить список процессов, запущенных в системе
- **kill**  
Завершение выполнения процесса
- **jobs**  
Отображение списка текущих заданий, выполняемых в фоновом режиме

- **fg**

Перемещение фонового процесса на передний план

- **ctrl + z**

Приостановите текущий процесс переднего плана и переместите его в фоновый режим

- **Control**

У нас есть довольно много контроля над запуском наших программ

## Задание.

Время для веселья

- Прежде всего, запустите несколько программ на своем рабочем столе. Затем используйте `ps`, чтобы идентифицировать их PID и убить их.
- Теперь посмотрите, можете ли вы сделать то же самое, но сначала переключитесь на другую виртуальную консоль.
- Наконец, поиграйте с командой `sleep` и перемещением процессов между передним и задним планом.