Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №2**

«Файловая система ОС UNIX»

Дисциплина: Операционные системы

Работу выполнил студент группы № 43501/1

Иванов А.А.

Работу принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Малышев И.А.

Санкт-Петербург

2018

# Часть 1

1. **Цель работы**

Изучение принципов организации файловой системы ОС UNIX на примере Linux.

1. **Теоретические данные**

В семействе операционных систем UNIX используется несколько типов файловых систем. Одной из наиболее распространенных является разработанная для UNIX System V файловая система s5 (system5, s51k b и.п.) Она характеризуется большой надежностью. Другая (BSD) отличается высоким быстродействием. В настоящее время появились новые типы файловых систем, отличающиеся повышенной надежностью, устойчивостью к программных сбоям, способностью работать в распределенной среде.

Файловые системы имеют иерархическую структуру, организуемую с помощью специальных файлов - каталогов. Файловая система физически располагается на так называемых блочных устройствах - носителе информации, хранение и обмен данными с которым производится поблочно, часто порциями по 1Kb. Само устройство не знает, что за файловая система находится на нем, оно позволяет работать с собой просто как с массивом блоков. Структуру файловой системы поддерживает ядро.

В UNIX не используется расширение имен для определения типов файлов (текстового, бинарного и т.п.). Существует один, универсальный тип «дискового» файла. В UNIX сняты ограничения на длину имени файла и постфикса (или суффикса).

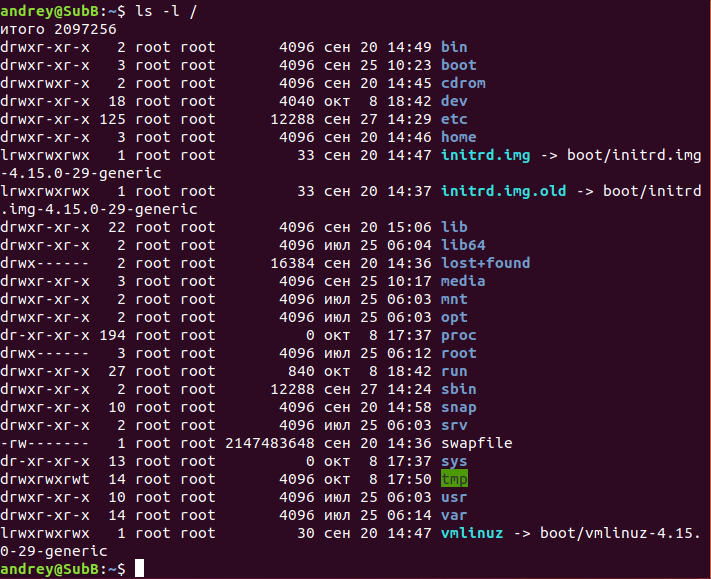
В UNIX имеется три вида файлов, доступ к которым идентичен:

* обычные дисковые файлы;
* каталоги;
* специальные файлы.

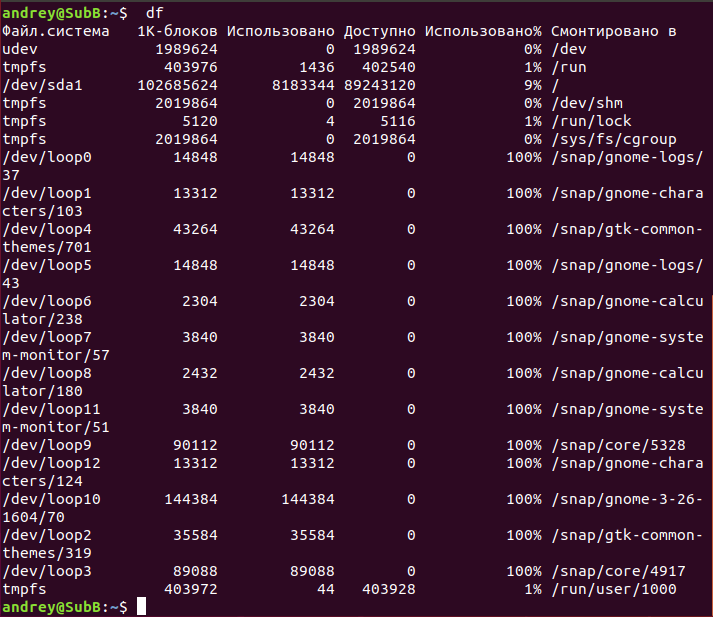
1. **Программа работы**. **Часть 1.**

**3.1 Иерархия каталогов**

Содержимое корня можно получить с помощью команды *ls -l /*:



Для получения информации о файловой системе воспользуемся утилитой *df*:

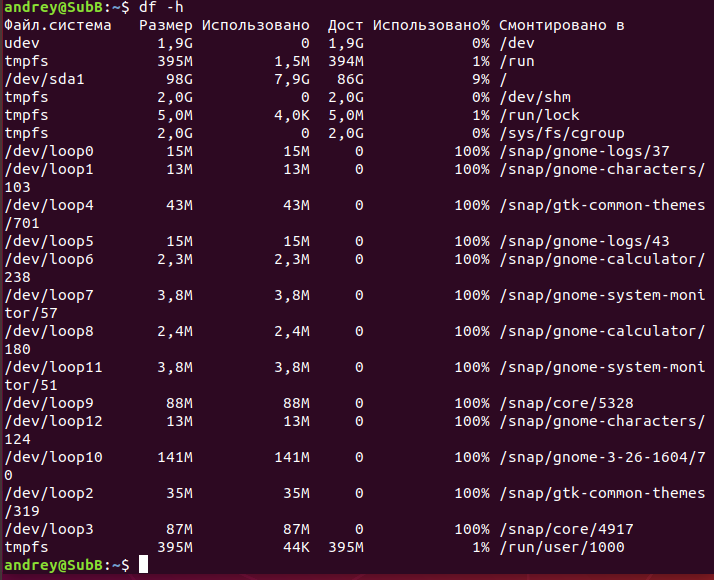


Размер блока равен:



Для определения размера блока использовалась команда stat, которая позволяет получить информацию о файловой системе или файлах. Размер блока равен 4096 Кб.

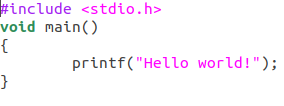
Определим связь логической структуры файловой системы и физической структурой, используя команду df:



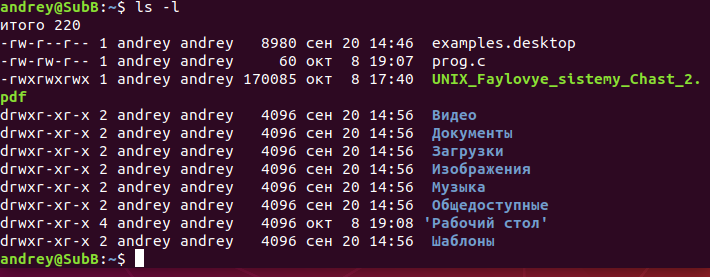
Команда df предназначена для получения информации о свободном дисковом пространстве, а так же выводит информацию о файловых системах. Ключ -h говорит выводить информацию в формате удобном для человека (указание размеров дискового пространства в Мб, Кб...).

**2.2. Невидимые символы**

Создаем файл prog.c с текстом программы:



Применив команду ls , определяем владельца и группу файла, права доступа, размер файла, время последнего изменения:



Владелец и группа – andrey

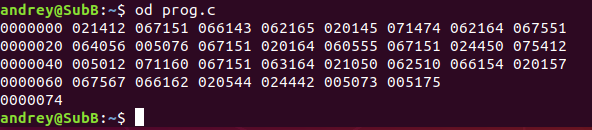
Права доступа:

* + владелец - запись и чтение;
  + группа – запись и чтение;
  + остальные - чтение.

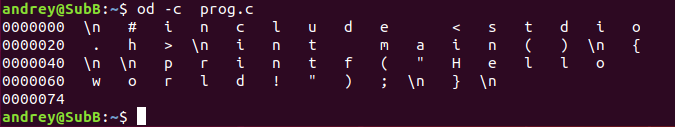
Размер файла = 6 байт

Время последнего изменения: 2018-10-8 19:07

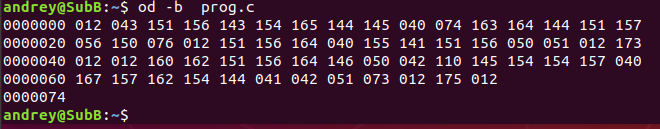
С помощью команды od (octal dump - восьмеричный дамп) выведим побайтную распечатку prog.c:



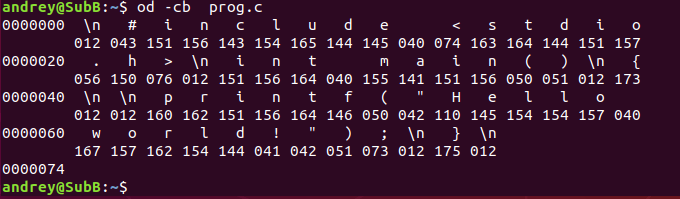
od -c выводит ASCII-символы.



od -b выводит побайтно коды символов в 8-миричной системе.



od –сb вывод ASCII-символы и символы 8-миричной виде.



Перевод строки - \n (код 012)

Пробел - код 040

Табуляция - \t (код 011)

**3.3. Ввод-вывод для файлов-терминалов**

С помощью tty определяем полное имя нашего файла-терминала:



Выведем информацию посредством echo, переназначив вывод на этот терминал:



Переназначим вывод на другой терминал:

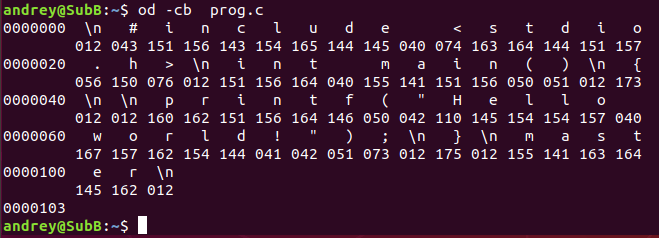


На консоли терминала pts/1 появляется andrey@SubB:~$ test Файл-терминал находится в каталоге /dev и воспринимается системой как файл-устрmойство. В результате, при выводе непосредственно в файл-терминал информация отображается в консоли.

Запустим cat без аргументов, переназначив ее вывод на файл prog.c и введем строчку:



После ввода команды терминал перешел в режим ожидания ввода символов. По нажатию Ctrl+D, ввод символов завершается, и данные записываются в файл. С помощью od убеждаемся, что в конец файла prog.c записывается введеная строчка (и символ перевода строки \n):

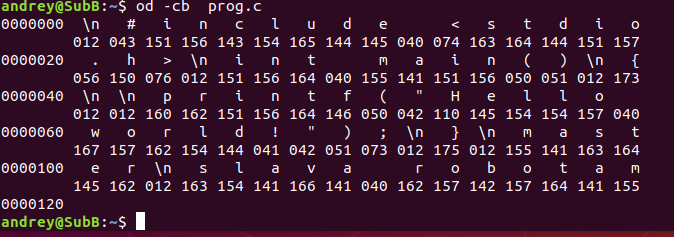


Запустим cat, наберем несколько символов без перевода строки:



Приглашения в shell появляется только после второго нажатия Ctrl+D. Это объясняется тем, что операционная система ищет в конце ввода символ перевода строки (\n) или конца потока ввода (символ Ctrl+D конец потока ввода). Если она его находит, то по нажатию Ctrl+D завершается ввод. Если не находит, то Ctrl+D воспринимается как конец потока ввода.

В файл записалась введенная строчка без символа перевода строки \n:



**3.4. Содержимое файлов**

Команда file проводит серию тестов, пытаясь классифицировать файлы, указанные в командной строке. Если обычный файл содержит текст в кодах ASCII, команда пытается по первому блоку угадать язык программирования. В том случае, когда обычный файл содержит двоичную информацию, file выясняет, является ли он объектным, библиотечным, выполнимым или каким-то иным. При запуске файла на исполнение (системный вызов exec) ОС проверяет его первой двухбайтное слово. Если оно содержит специальной значение («магическое число»), ОС считает такой файл содержащим машинные коды и запускает эту программу.

С помощью команды file определим содержимое файла prog.c:



Исследуем с помощью этой команды различные файлы:

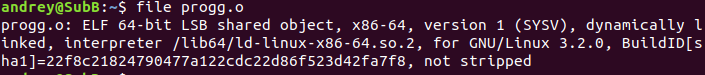
1) /bin



2) текстовый файл



3) объектный файл

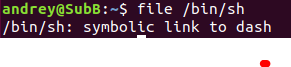


4) исполняемый файл



ELF - формат исполняемых и компонуемых файлов. LSB - стандарт на исполняемые файлы. x86-64 - процессор для которого собран файл. dynamically linked (uses shared libs) - собран. GNU/Linux 3.2.0 - операционная система. not stripped - включена отладочная информация.

5) /bin/sh - ссылка на dash (интерпретатор)



6) /etc/passwd - файл паролей, по содержимому которого определяется соответствие между номерами и именами владельцев.



7) /lib/libc.a - не существует такой библиотеки.



С помощью tail запишем хвост /bin/sh (15 последних символов) в файл hhh:



Исследуем его с помощью file. Так как команда file распознает вид файла по первому блоку, а в файл hhh записан хвост, в данной ситуации вид файла распознать не удается:



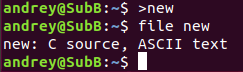
С помощью od определим магическое число (первые два байта) executable файлов:



Отсюда видно, что магическое число (первые 2 байта): 177 105

Сравним полученные результаты с содержимым файла /etc/magic, содержащим магические числа системы. В этом файле нет магического числа.

Возьмём содержимое файла prog.c и запишем в файл new (без расширения). В результате команда file распознаёт его также, как и исходный файл



**3.5. Права доступа**

**3.5.1 Права доступа процессов к файлам**

Категории пользователей для определения прав доступа: владелец, группа, остальные.

Виды прав доступа: чтение (r), запись (w) и исполнение (x).

Определим права доступа к своему "домашнему" каталогу:



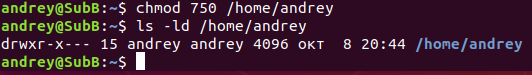
Владелец - andrey, группа - andrey.

Для владельца - чтение, запись и исполнение (rwx).

Для группы - чтение и исполнение (r-x).

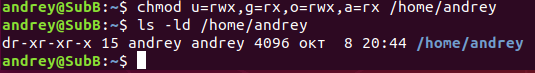
Для остальных - исполнение (x).

Посредством команды chmod, обеспечим все права себе, возможность чтения и поиска для группы и никаких прав остальным:



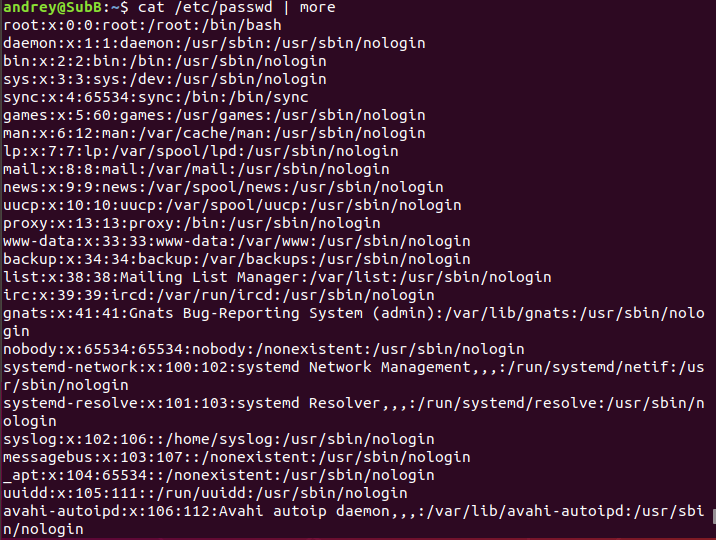
Права доступа расположены в следующем порядке: чтение, запись, исполнение. Таким образом мы должны установить единицы в тех разрядах, которые соответствуют нужным нам правам. Первая опция (7=111) задает все права доступа для владельца, вторая (5=101) - права чтения и поиска для группы, а третья(0=000) не обеспечивает правами остальных.

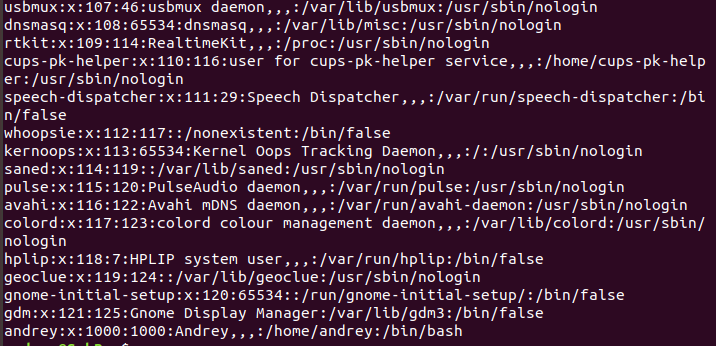
Команде chmod можно задавать параметры и другим способом: указанием кому изменить права доступа (классы пользователей — u, g, o, a):



**3.5.2 Файл паролей /etc/passwd**

Cтруктура файла паролей /etc/passwd:





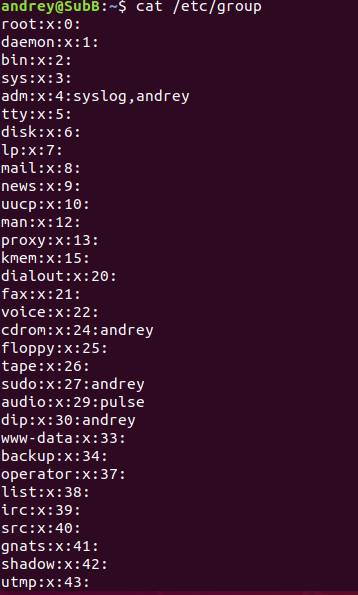
Назначение полей файла /etc/passwd:

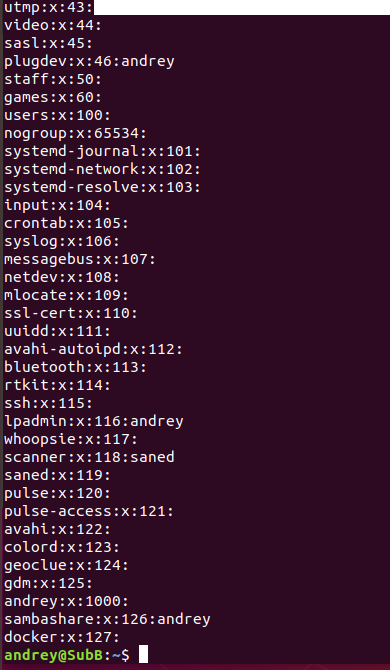
*регистрационное имя или логин* : *хэш пароля* : *идентификатор пользователя* : *идентификатор группы* : *информационное поле GECOS* : *домашний каталог* : *какой shell использует по умолчанию*

Поле GECOS хранит вспомогательную информацию о пользователе (полное имя, адрес, рабочий телефон, домашний телефон и т.д.).

Группы тоже имеют имена, информация об этом содержится в файле /etc/group.

Структура файла /etc/group:



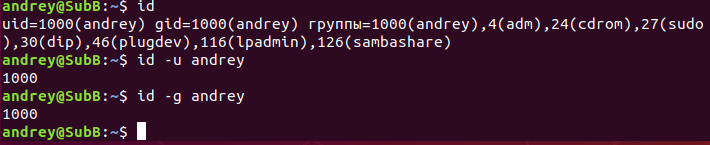


Назначение полей файла /etc/group:

*имя группы* : *пароль* : *идентификатор группы* : *список пользователей в группе*

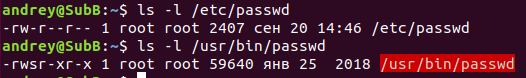
Если в файле /etc/passwd у пользователя указана группа в качестве основной, то в файле /etc/group для этой группы данный пользователь не указывается.

Посредством команды id, определим имена и идентификаторы владельца и группы нашего текущего процесса (shell`а). За вывод идентификаторов отвечают параметры -u (владелец) и -g (группа):



**3.5.3.Переустановка идентификатора владельца процесса**

С помощью ls определим права доступа к файлу /etc/passwd, и к файлу /usr/bin/passwd (программа для смены пароля):



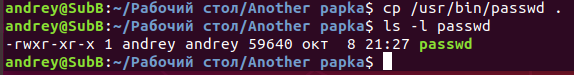
«s» соответствует установленному биту SUID. При запуске обычного исполняемого файла, он получает реальные идентификаторы пользователя, его запустившего. В большинстве случаев эффективные идентификаторы такие же как реальные и по этим идентификаторам определяются права доступа процесса. В результате права доступа процесса чаще всего определяются правами пользователя, запустившего процесс.

Если на исполняемый файл установлен бит SUID, то при выполнении эта программа автоматически получит эффективные идентификаторы владельца исполняемого файла. То есть, независимо от того, кто запускает эту программу, она при выполнении имеет права хозяина этого файла.

У утилиты passwd установлен бит SUID из соображений, что пользователь может менять себе пароль самостоятельно, но чтобы это сделать необходимо обладать правами root (так как право на изменение файла /etc/passwd есть только у root).

сз

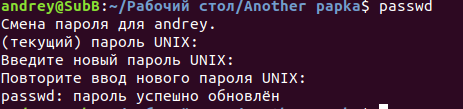
Скопируем к себе /usr/bin/passwd с тем же именем. Проверим результат, в т.ч. владельца копии:



При сравнении исходного и скопированного файла было обнаружено, что они одинаковые:



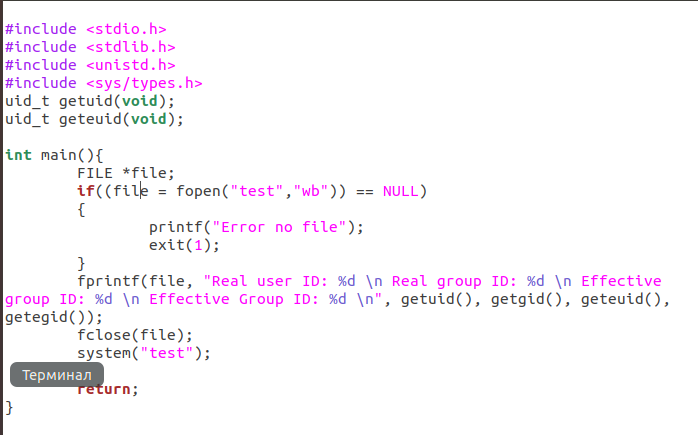
Попробуем изменить себе пароль с помощью собственной копии passwd:



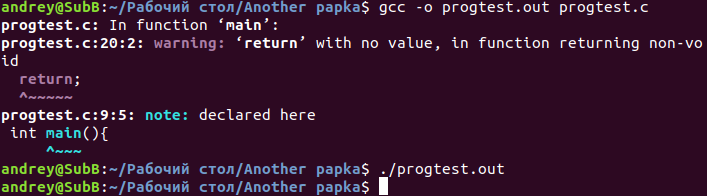
Пароль изменился, потому что идентификатор владельца (andrey andrey) имеет права, аналогичные root root.

Из всех вышеприведенных операций следует, что права доступа для исполняемых файлов определяются эффективными идентификаторами.

Создадим С-программу, которая выводит данные в файл:



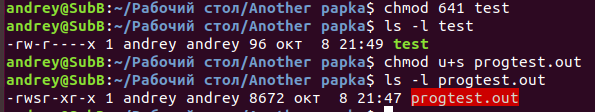
С помощью системных вызовов getuid(), getgid(), getegid(), geteuid() занесем в файл информацию о реальных и эффективных идентификаторах пользователя. Создадим исполняемый файл и выполним программу:



В выходной файл test записалась информация об ID:



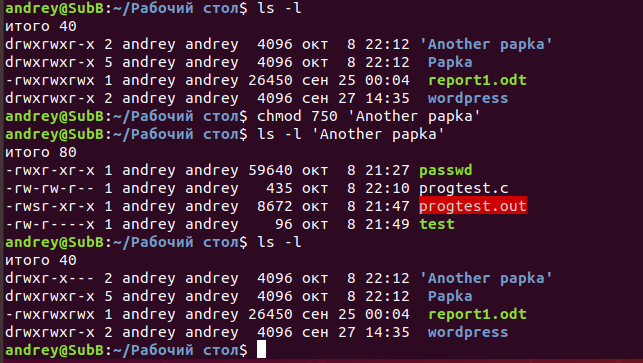
С помощью назначения соответствующих прав доступа обеспечим режим обращения к файлу с данными только через нашу программу и установим для исполняемого файла флаг SUID:



После проделанных операций записывать информацию в test может только andrey. Владельцем программы prog.out также является andrey. Таким образом, если другой пользователь запустит программу prog.out, то увидит информацию, записываемую в test, при этом не имея права что либо записывать в этот файл и читать его.

**3.5.4. Права доступа к каталогам**

Определим права доступа к своему «домашнему» каталогу. Обеспечим посредством команды chmod все права себе, возможность чтения и поиска для нашей группы и никаких прав остальным:



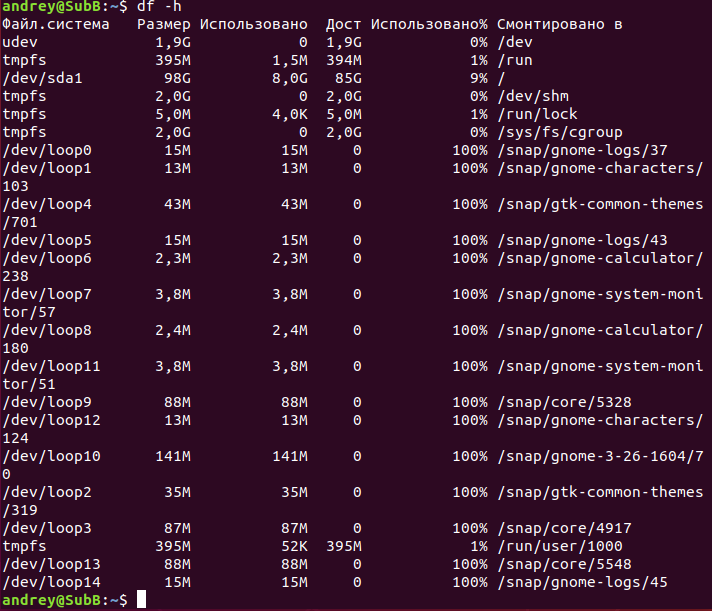
Попытаемся поменять владельца файла test с помощью chown:



Попытка не удалась, так как администратором не предоставлено прав на изменение владельцев файлов.

**3.6. Содержимое каталогов**

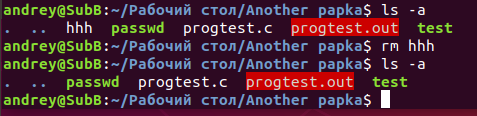
С помощью команды df определим занятое, свободное и общее пространство на дисках:



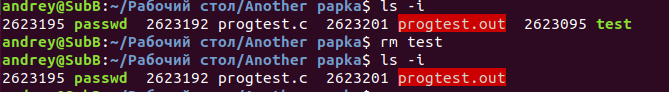
Посредством od прочтем содержимое любого каталога как файла (od -cb):



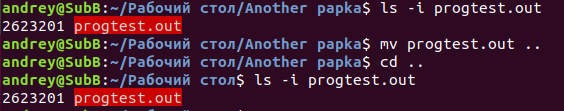
Удалим файл и посмотрим содержимое каталога с помощью ls -a:

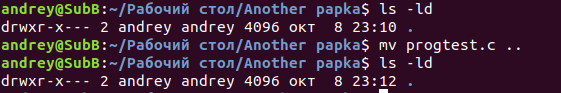


Удалим файл hhh и посмотрим содержимое каталога с помощью ls -i. По результатам видно, что i-node файлов не изменились:



Переместим файл prog.out. Как и в случае с удалением, i-node файла prog.out не изменился:





Размер файла-каталога не изменился. Система не сжимает каталог после уменьшения в нем числа файлов.

Создадим жесткую ссылку на файл prog.c:



ls показывает, что на файл prog.c есть две жесткие ссылки.



Оба файла (prog.c и prog1.c) имеют один i-node (2623192). Это объясняется тем, что на самом деле существует всего один файл, но две ссылки, которые отображаются как отдельные файлы.

Найдем в /bin или /usr/bin редактор vi и определим количество связей к нему:



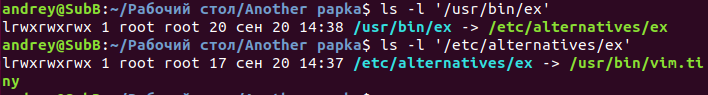
Файл /usr/bin/vi — мягкая ссылка на /etc/alternatives/vi



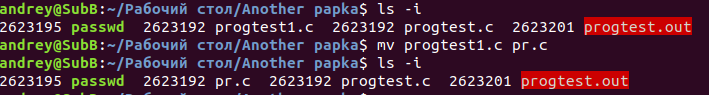
Файл /etc/alternatives/vi — мягкая ссылка на /usr/bin/vim.tiny



На этот файл существует только одна ссылка. Аналогично файл /usr/bin/ex ссылается на /usr/bin/vim.tiny. Файлы /usr/bin/vi и /usr/bin/ex занимают 20 байт.



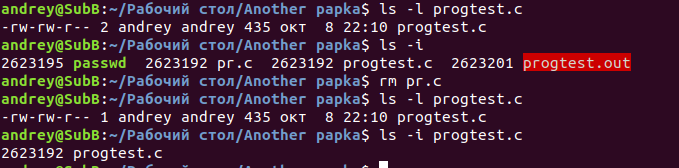
Переименуем файл prog1.c (mv):



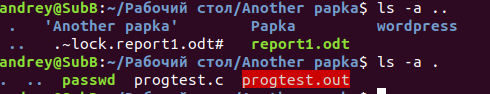
Номер описателя файла pr.c не изменился (2623192). Это объясняется тем, что во время переименования не было создано никаких новых файлов, и они не перемещались, а изменился лишь атрибут файла — имя файла.

Точно так же дескриптор файла не меняется при перемещении файлов между каталогами. При копировании файлов он изменяется.

Удалим дополнительную связь к prog.c и посмотрим, как изменится его i-node. В результате i-node остался прежним, а количество ссылок уменьшилось:



Во всех каталогах существуют имена . и .., которые обозначают сам каталог и родительский каталог.



**3.7. Специальные файлы**



При выводе посредством команды ls -l для файлов указывается тип (первый символ) и нумерация (второй символ). В отличие от обычных файлов, специальные файлы устройств в действительности есть только указатели на соответствующие драйверы устройств в ядре.

1. **Выводы**

При выполнении данной лабораторной работы были изучены принципы организации файловой системы ОС UNIX на примере Linux:

* Классическая файловая система представляет данные в виде вложенных друг в друга **каталогов** (их ещё называют папками), в которых содержатся **файлы**. Один из каталогов является «вершиной» файловой системы («корнем»), в нём содержатся все остальные каталоги и файлы.
* В Linux корневой каталог называется «/». Полные имена всех остальных каталогов получаются из «/», к которому дописываются справа имена последовательно вложенных друг в друга каталогов.
* Корневой каталог в Linux всегда только один, все остальные каталоги вложены в него. Можно говорить, что для пользователя файловая система представляет собой единое целое, однако разные части файловой системы могут находиться на совершенно разных устройствах: разных разделах жёсткого диска, на разнообразных съёмных носителях. Для того чтобы соорудить единое дерево с одним корнем, используется процедура **монтирования**.
* Монтирование — это подключение в один из каталогов целой файловой системы, находящейся где-то на другом устройстве. Эту операцию можно представить как «прививание» ветки к дереву. Для монтирования необходим пустой каталог — он называется **точкой монтирования**. Происходит объявление, что в данном каталоге (пока пустом) нужно отображать файловую систему, доступную на таком-то устройстве или же по сети. После этой операции в каталоге (точке монтирования) появятся все те файлы и каталоги, которые находятся на соответствующем устройстве.
* Для Linux самой важной является **корневая файловая система** (root filesystem). Именно к ней затем будут подключаться все остальные файловые системы на других устройствах. Точкой монтирования служит «/».

Так же были получены сведения об индексном дескрипторе (inode). Это структура данных в файловых системах, в которой хранится вся информация о стандартных файлах, каталогах или других объектах файловой системы, кроме непосредственно данных и имени.

При создании файловой системы создаются также и структуры данных, содержащие информацию о файлах. Каждый файл имеет свой индексный дескриптор, идентифицируемый по уникальному номеру, в файловой системе, в которой располагается сам файл. Индексные дескрипторы хранят информацию о файлах, такую как принадлежность владельцу (пользователю и группе), режим доступа (чтение, запись, запуск на выполнение) и тип файла.

Подобная концепция играет важную роль при восстановлении поврежденных файловых систем.

* Номер индексного дескриптора заносится в таблицу индексных дескрипторов в определенном месте устройства; по номеру индексного дескриптора ядро системы может считать содержимое инода.
* Номер индексного дескриптора файла можно посмотреть используя команду ls –i.

## Часть 2

# 1. Программа работы

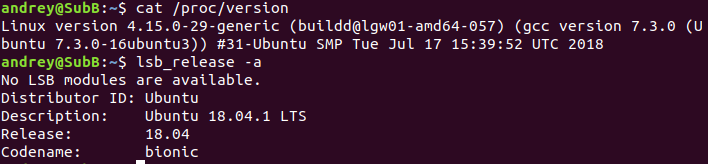
1. Используя команду ls, получить информацию о всех возможных типах файлов ФС в выходном файле. Оформить в виде скрипта.
2. Получить все жесткие ссылки на заданный файл, находящиеся в разных каталогах пользовательского пространства (использовать конвейер команд и фильтрацию).
3. Изучить команду find. Используя ее ключи, получить расширенную информацию о типах файлов.
4. Проанализировать содержимое файла-каталога с помощью команды od при выполнении различных операций с файлами:  
   а) организовать пустой каталог  
   б) скопировать в него файл  
   в) переименовать  
   г) удалить  
   д) создать подкаталог  
   е) создать в нем файл  
   ж) сравнить с UNIX ФС

Последовательность выполнения операций над файлами и каталогами, а также сравнение результатов в файле-каталоге задать в виде командного файла, позволяющего автоматизировать процесс анализа информации в файле-каталоге при внесении в него изменений при любой операции.

1. Ознакомиться с содержимым файла /etc/passwd, /etc/shadow, утилитой /usr/bin/passwd, проанализировать права доступа к этим файлам
2. Изучить команды изменения прав доступа и владельца chmod, chown на примере специально созданного для этих целей каталога с файлами. Ознакомиться с флагом SUID. Экспериментально установить, как формируются итоговые права на использование файла, если права пользователя и группы, в которую он входит, различны.
3. Разработать программу-шлюз для доступа к данным другого пользователя при отсутствии прав на чтение информации (используя флаг SUID)
4. Утилитами df, fstab получить информацию о ФС и ОС, установленных на компьютере.
5. Проанализировать все возможные способы формирования ссылок (ln, link, cp и т.д.) и подтвердить экспериментально. Предложить скрипт, подсчитывающий и перечисляющий все символические ссылки на файл в разных каталогах.
6. Пояснить принцип работы утилиты file. Определить, где находятся магические числа, идентифицирующие тип файла, на примере исполняемого файла и любого другого типа по выбору. Команду file выполнить с разными ключами.
7. Сопоставить возможности исполнения наиболее часто используемых операций, варьируя правами доступа к файлу и каталогу (оформить в виде таблицы).

# 2. Ход работы

Узнаем систему и версию ядра, а также информацию о дистрибутиве:



## 2.1. Типы файлов

В Unix-подобных системах все объекты — это файлы. Все объекты — это весьма разнообразное явление. Поэтому тип файла в Linux – это скорее тип объекта, но не тип данных в файле.

|  |  |  |  |
| --- | --- | --- | --- |
| Типы файлов | | Описание | Назначение |
| Обычные файлы (regular file) | - | текст, изображения, бинарные файлы, библиотеки и т.д. | хранение символьных и двоичных данных |
| Каталоги (directory) | d |  | организация доступа к файлам |
| Символьные ссылки (soft link) | l | является указателем на оригинальный файл | предоставление доступа к файлам, расположенных на любых носителях |
| Блочные устройства | b |  | предоставление интерфейса для взаимодействия с аппаратным обеспечением компьютера |
| Символьные устройства | c |
| Каналы (pipe) | p |  | организация взаимодействия процессов в операционной системе |
| Сокеты (socket) | s |  |

Исследуем структуру файловой системы на ноутбуке и назначение содержимого корневого каталога. Реализуем это двумя путями: через командную строку и с помощью скрипта.

Обычный файл:



Директория:



Символьная ссылка:



Блочные устройства:



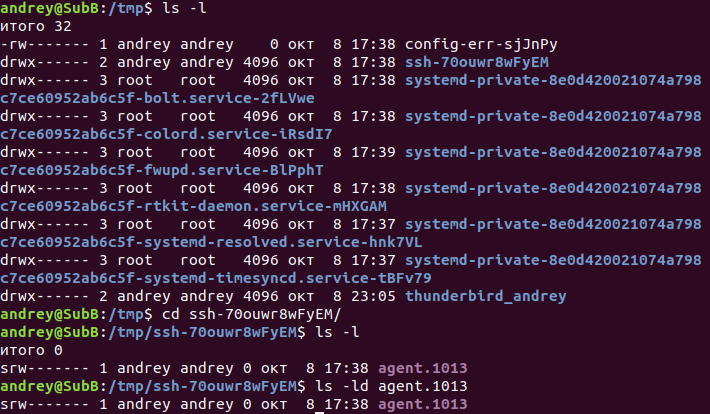
Символьные устройства:



Каналы:



Сокеты:



Реализация через одну команду:

С помощью утилиты ls с флагом R запросим список всех файлов из корневого каталога (“/”), уберем строки, начинающиеся с «итого» и строки, начинающиеся со слэша, возьмем только первый символ каждой строки (cut –c1), а затем уберем дубликаты (sort –u), и вместо перехода на новую строку сделаем пробел (tr ‘\n’ ‘ ‘).



Разбор команды:

• sudo – необходимо для получения доступа

• ls / – указываем корневой каталог

• -lR – устанавливаем вывод полной информации и рекурсивное прохождение

• 2>/dev/null – перенаправление потока ошибок

• grep -v '^итого' – избавление от лишней информации

• grep -v ^/ – избавление от лишней информации

• cut –c1 – выбираем из каждой строки файла первый символ

• sort -u – сортируем и скрываем одинаковые объекты

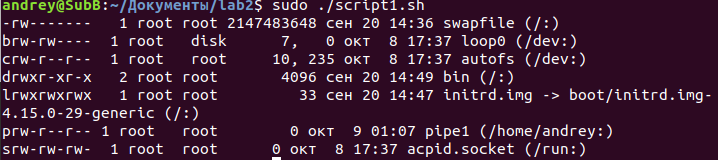
• tr '\n' ' ' – заменяем переводы строк на пробелы

Реализация через скрипт:

|  |
| --- |
| #!/bin/bash  # echo "Searching file types in system stared!"  results=$(ls / -lR 2>/dev/null 1>/home/andrey/Документы/lab2/founded\_files.txt)  types=$(cat /home/andrey/Документы/lab2/founded\_files.txt | grep -v ^итого|grep -v ^/ | cut -c1 | sort -u | tr '\n' ' ')  for i in $types  do  type\_filepath=''  type\_file=''  cat /home/andrey/Документы/lab2/founded\_files.txt | grep "^\($i\|\/\)" | while read line  do  if [[ $line =~ ^/ ]] #Определяем: путь или файл? Поиск по шаблону: начало строки – это /  then  type\_filepath=$line  else  type\_file=$line  echo "$type\_file ($type\_filepath)"  break #Нашли, что надо: покидаем цикл.  fi  done;  done |

Разбор скрипта:  
Скрипт запрашивает рекsriptурсивный список всех файлов, сохраняет результат в файл, затем ищет в нем все имеющиеся типы файлов. После этого для каждого имеющегося типа файла выводит пример файла заданного типа и путь до него в скобочках в конце.

Результат исполнения скрипта: выведены примеры на каждый тип файлов, а в конце в скобках указывается путь к этому файлу.

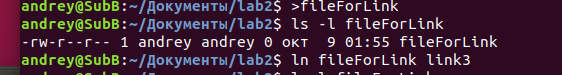


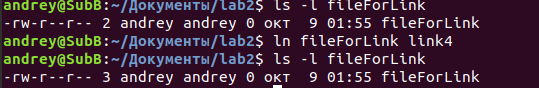
## 2.2. Получение жестких ссылок

Получить все жесткие ссылки на заданный файл, находящиеся в разных каталогах пользовательского пространства (разными способами, не применяя утилиты file и find). Использовать конвейеризацию и фильтрацию. Оформить в виде скрипта.

При создании файловой системы создаются и структуры данных, содержащие информацию о файлах. Каждый файл имеет свой уникальный номер – inode (индексный дескриптор). Сначала определим inode указанного файла, затем получим все файлы и сравним их inode. Таким образом, получим данные об искомых файлах.

Создаём все нужные файлы и ссылки:

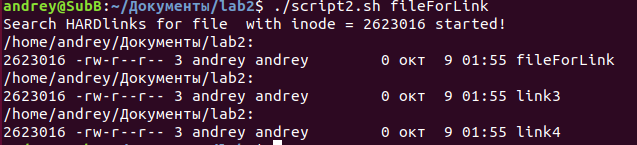




Реализация через скрипт:

|  |
| --- |
| #!/bin/bash  IFS=$'\n' #Разделитель полей во вводимой строке  inode=$(ls $1 -li | cut -d' ' -f 1)  echo "Search HARDlinks for file $path with inode = $inode started!"  inode\_results=$(ls / -liR 2>/dev/null | grep "^\($inode\|\/\)" > $HOME/inode\_results.txt)  filepath=''  mode=0;  for line in $(cat $HOME/inode\_results.txt)  do  if [[ $line =~ ^/ ]] #Поиск по шаблону: начало строки – это /  then  filepath=$line  mode=0;  else  echo $filepath  mode=1;  if [ $mode -eq 1 ]  then  echo "$line"  else  echo "error"  fi  fi  done |

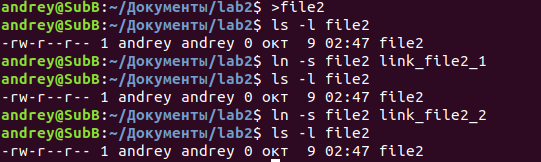
Результат работы скрипта: кол-во жестких ссылок на файл fileForLink 3, как и должно быть.



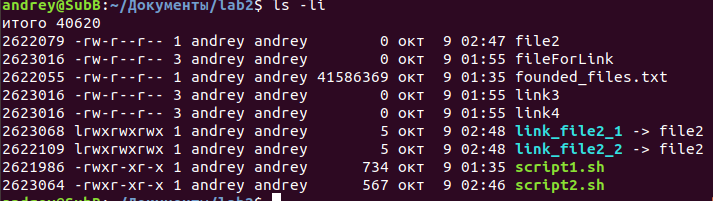
## 2.3. Формирование символьных ссылок

Проанализировать все возможные способы формирования *символьных ссылок* (ln, link, cp и т.д.), продемонстрировать их экспериментально. Предложить скрипт, подсчитывающий и перечисляющий все полноименные символьные ссылки на файл, размещаемые в разных местах файлового дерева.

### 2.3.1. ln

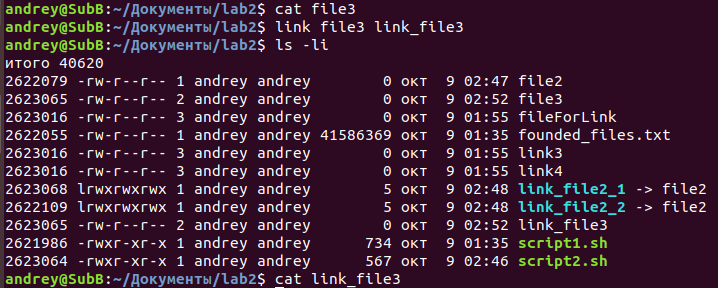


Кол-во жестких ссылок на файл не изменилось. Посмотрим через команду «ls -li»



### 2.3.2. link

Пробуем создать символьную ссылку с помощью команды link

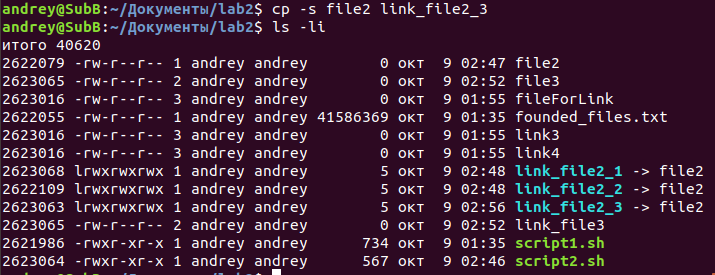


Создали ЖЁСТКУЮ ссылку: если удалить/переименовать file3, то символьная ссылка перестанет работать, а link\_file3 – нет (будет выдавать ту же информацию, что и раньше).

При создании символьной ссылки через ln и cp (см.далее п.2.3.3) счётчик ссылок на файл не увеличивается. При использовании link – увеличивается (как при создании жёстких ссылок).

### 2.3.3. cp

Пробуем создать символьную ссылку с помощью команды cp с параметром –s (создать символьную ссылку вместо копирования каталога):



### 2.3.4. Реализация через скрипт

Далее представлен скрипт, подсчитывающий и перечисляющий все полноименные символьные ссылки на файл, размещаемые в разных местах файлового дерева.

Скрипт подсчитывает количество символьных ссылок, а также вывод их местоположения относительно текущего каталога.

|  |
| --- |
| #!/bin/bash  echo "Symbolic links count:" # Вывод сообщения в консоль  ls -lR | grep '\-> '$1 | wc –l # Рекурсивный поиск файлов, в которых присутствует  # ссылка на введенный пользователем файл, подсчет  # полученных строк  echo "Links:" # Вывод сообщения в консоль  ls $HOME -lR 2>/dev/null| awk '{if($0~/^\//) path=substr($0,0,length($0)-1); else { if ($0~/^l/) $(NF-2)=path"/"$(NF-2);} print $0} ' |grep '\-> '$1  # Рекурсивный поиск с выводом ошибок в /dev/null  # Если строчка начинается с /(директория)  # Сохраняем путь текущей директории в переменную path  # Иначе, если строка начинается с символа l  # Заменяем путь в NF-2 столбце и выводим результат  # Фильтруем только ссылки на введенный файл |

## 2.4. Получение символьных ссылок без утилиты file

Получить все символьные ссылки на заданный в качестве входного параметра файл, не используя file (разными способами, не применяя утилиту file).

|  |
| --- |
| #!/bin/bash  echo "Number of symbol links:" #Выводим кол-во строк  ls / -lR 2>/dev/null| grep '\-> '$1 | wc -l  echo "Links:" #Выводим ссылки  ls -liR /home | grep '\-> '$1 |

Результат исполнения скрипта:

## 

## 2.5. Утилита find

Изучить утилиту find, используя ее ключи получить расширенную информацию о всех типах файлов. Создать примеры вложенных команд.

Синтаксис утилиты:  
find [путь] [опции] [критерии поиска] [действия над файлами]

Описание:  
find — утилита поиска файлов по имени и другим свойствам, используемая в UNIX‐подобных операционных системах. Может производить поиск в одном или нескольких каталогах с использованием критериев, заданных пользователем. По умолчанию, find возвращает все файлы в рабочей директории. Более того, find позволяет применять пользователю определённые действия ко всем найденным файлам. Также поддерживаются регулярные выражения.

Пример:

find / -name config -print

Разбор команды:

* искать начиная с каталога / (первый параметр команды find)
* искать файлы с именем config (-name config)
* выводить имена файлов на экран (-print).

Поиск рекурсивный (то есть при поиске просматриваются все вложенные директории).

Ключи команды find:

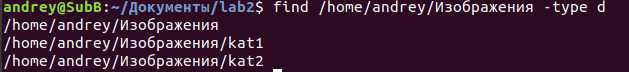
* -name — искать по имени файла, при использовании подстановочных образцов параметр заключается в кавычки. Опция `-name' различает прописные и строчные буквы; чтобы использовать поиск без этих различий, воспользуйтесь опцией `-iname'.
* -type — тип искомого: f=файл, d=каталог, l=ссылка (link), p=канал (pipe), s=сокет.
* -user — владелец: имя пользователя или UID.
* -group — владелец: группа пользователя или GID.
* -perm — указываются права доступа.
* -size — размер: указывается в 512-байтных блоках или байтах (признак байтов — символ «c» за числом).
* -atime — время последнего обращения к файлу (в днях).
* -amin — время последнего обращения к файлу (в минутах).
* -ctime — время последнего изменения владельца или прав доступа к файлу (в днях).
* -cmin — время последнего изменения владельца или прав доступа к файлу (в минутах).
* -mtime — время последнего изменения файла (в днях).
* -mmin — время последнего изменения файла (в минутах).
* -newer другой\_файл — искать файлы созданные позже, чем другой\_файл.
* -delete — удалять найденные файлы.
* -ls — генерирует вывод как команда ls -dgils.
* -print — показывает на экране найденные файлы.
* -print0 — выводит путь к текущему файлу на стандартный вывод, за которым следует символ ASCII NULL (код символа 0).
* -exec command {} \; — выполняет над найденным файлом указанную команду; обратите внимание на синтаксис.
* -ok — перед выполнением команды указанной в -exec, выдаёт запрос.
* -depth или -d — начинать поиск с самых глубоких уровней вложенности, а не с корня каталога.
* -maxdepth — максимальный уровень вложенности для поиска. «-maxdepth 0» ограничивает поиск текущим каталогом.
* -prune — используется, когда вы хотите исключить из поиска определённые каталоги.
* -mount или -xdev — не переходить на другие файловые системы.
* -regex — искать по имени файла используя регулярные выражения.
* -regextype тип — указание типа используемых регулярных выражений.
* -P — не разворачивать символические ссылки (поведение по умолчанию).
* -L — разворачивать символические ссылки.
* -empty — только пустые каталоги.

Примеры:

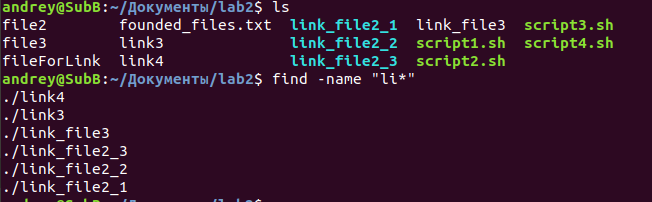
1. Найти файл с именем script1.sh, начиная поиск с каталога /home/andrey

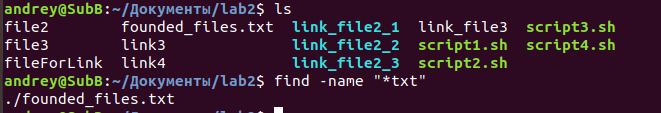


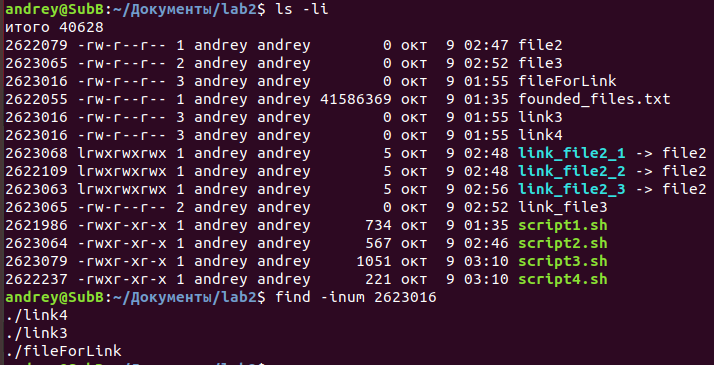
1. Вывести все файлы(каталоги) с типом d относительно текущей директории.



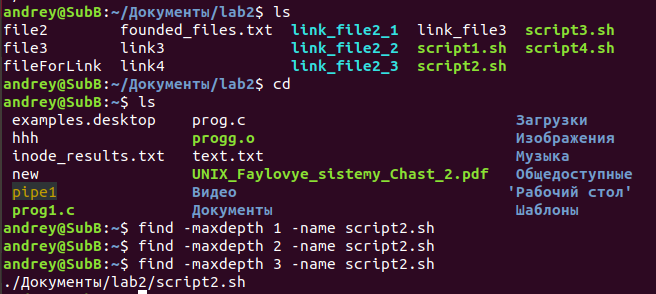
1. Вывести все файлы, которые начинаются с символов la



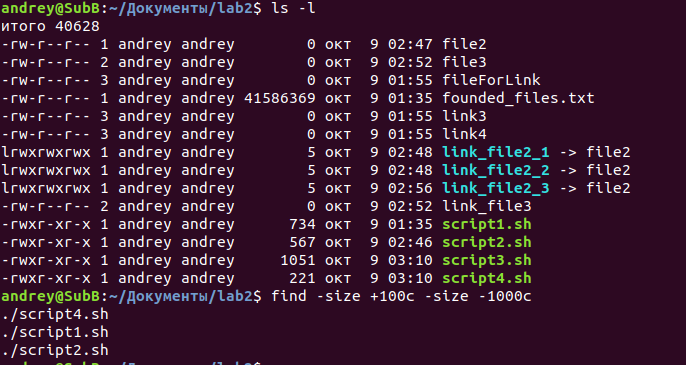
1. Вывести все файлы с расширением .txt
2. Вывести все файлы с inode = 316034



1. Найти файл с именем , причем глубина вложенности задана отдельным параметром maxdepth



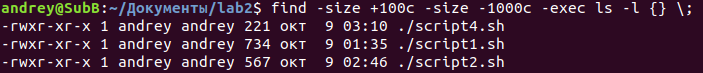
1. Вывести все файлы, размеры которых больше 100, но меньше 1000 байт.

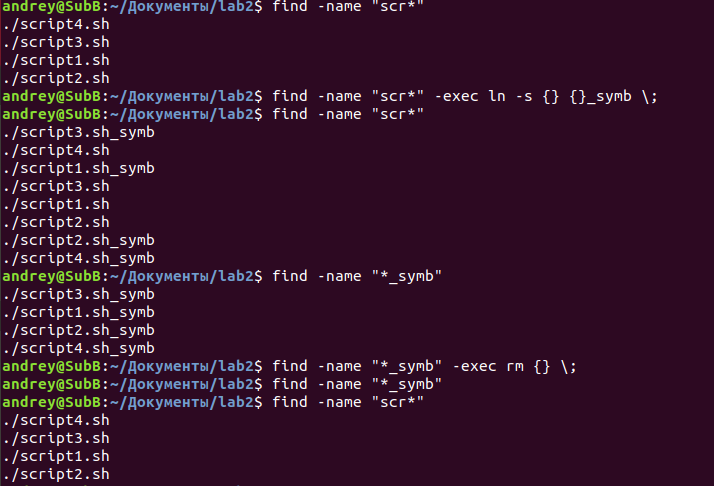


Использование утилиты find с опцией exec позволяет создавать вложенные команды. Вызов exec будет происходить из текущей директории. Все символы, следующие за командой exec, считаются ее аргументами до символа ";". Аргумент "\{\}" заменяется на имя рассматриваемого файла каждый раз, когда он встречается среди аргументов команды.

Пример:

1. С помощью find были найдены файлы, размеры которых которые больше 100, но меньше 1000 байт, затем используя ls была выведена их более подробная информация.



1. С помощью find были найдены файлы, начинающиеся на la, были созданы символьные ссылки на эти файлы, затем удалены ранее созданные символьные ссылки.

## 2.6. Утилиты od и \*dump

Проанализировать содержимое заголовка файла, а также файла-каталога с помощью утилит od и \*dump. Если доступ к файлу-каталогу возможен (для отдельных модификаций POSIX - совместимых ОС), проанализировать изменение его содержимого при различных операциях над элементами, входящими в его состав (файлами и подкаталогами).

### 2.6.1. od

Синтаксис команды od:

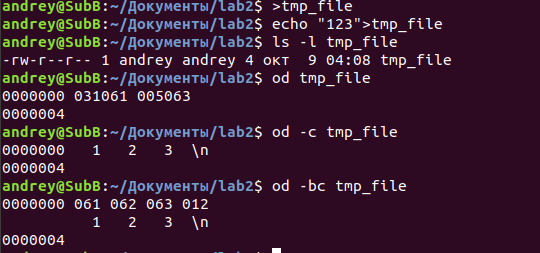
od [- опции] [-A в каком формате будет колонка позиций ] [-j начать с какого байта ] [-N длина ] [-t формат вывода ] [файл]

Опции:

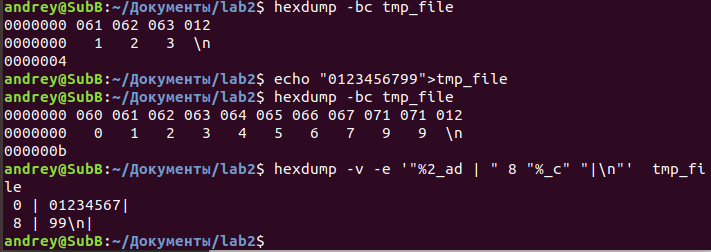
* -b Выдавать байты в восьмеричном виде.
* -c Интерпретировать байты как коды ASCII. Некоторые служебные символы выводятся по соглашениям, принятым в языке C: пусто=\0, забой=\b, переход\_к\_новой странице=\f, перевод\_строки=\n, возврат\_каретки=\r, табуляция=\t; все остальные выводятся как восьмеричные числа из 3 цифр.
* -d Выдавать 16-разрядные слова в виде десятичных чисел без знака.
* -o Выдавать 16-разрядные слова в восьмеричном виде.
* -s Выдавать 16-разрядные слова в виде десятичных чисел со знаком.
* -x Выдавать 16-разрядные слова в шестнадцатеричном виде.

Утилита od позволяет произвести вывод содержимого файла в определенном формате. В выводе утилиты od каждая строка содержит выходное смещение и следующий за ним блок данных. По умолчанию смещение выводится в восьмеричном формате.

Пример использования:



### 2.6.2. hexdump



Аналогично утилите od было выведено содержимое файла с указанием символов и их восьмиричных кодов. Далее этот же файл был выведен с использованием флага -e для установления формата вывода. Сперва было выведено смещение относительно начала файла, а затем установленное количество символов для вывода.

### 2.6.3. objdump

Утилита objdump позволяет получить информацию по содержимому объектного файла:

## 

## 2.7. Записи в каталогах

Определить максимальное количество записей в каталоге. Изменить размер каталога, варьируя количество записей (для этого создать программу, порождающую новые файлы и каталоги, а затем удаляющую их, предусмотрев промежуточный и конечный вывод информации о размере подопытного каталога).

Создадим скрипт для генерации файлов до тех пор, пока каталог не изменит свой размер.Скрипт наполняет каталог другими каталогами, до тех пор пока размер родительского каталога не изменится. После создания каждой папки будем считывать размер файла-директории. Размер считываем с помощью команды du –s (определяем занимаемое дисковое пространство).

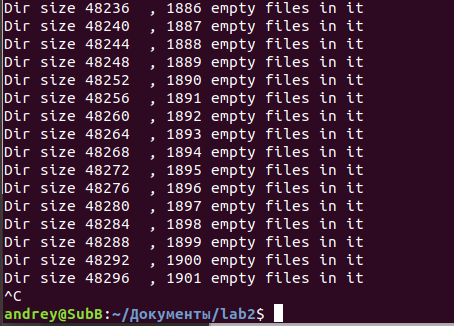
|  |
| --- |
| #!/bin/bash  mkdir /home/andrey/Документы/lab2/TestDir #Создаем тестовый каталог  old\_size=$(ls -l /home/andrey/Документы/lab2/TestDir/ | du -s | cut -b 1-6) #Сохраняем его размер в переменную  new\_size=$old\_size #Копируем его размер в другую переменную, предназначенную для нового размера  i=0 #Задаем начальное значение счетчика цикла  #Условие выхода из цикла: совпадение размеров  while [ "$old\_size" == "$new\_size" ]  do  mkdir /home/andrey/Документы/lab2/TestDir/$i #создаем файл-каталог внутри тестового каталога  let "i=$i+1" #увеличиваем счетчик цикла  new\_size=$(ls -l /home/andrey/Документы/lab2/TestDir/ | du -s | cut -b 1-6) #определяем новый размер файла  done  echo "Dir size changed from $old\_size to $new\_size when $i empty files were created in it" #Выводим результат |

Выполним скрипт. После создания в нем 1 пустого каталога, его размер увеличился на 4 КБ.

Уберем ограничение на кол-во создаваемых каталогов:

|  |
| --- |
| #!/bin/bash  mkdir /home/andrey/Документы/lab2/TestDir #Создаем тестовый каталог  old\_size=$(ls -l /home/andrey/Документы/lab2/TestDir/ | du -s | cut -b 1-6) #Сохраняем его размер в переменную  new\_size=$old\_size #Копируем его размер в другую переменную, предназначенную для нового размера  i=0 #Задаем начальное значение счетчика цикла  #Условие выхода из цикла: совпадение размеров  while [1]  do  mkdir /home/andrey/Документы/lab2/TestDir/$i #создаем файл-каталог внутри тестового #каталога  let "i=$i+1" #увеличиваем счетчик цикла  new\_size=$(ls -l /home/andrey/Документы/lab2/TestDir/ | du -s | cut -b 1-6) #определяем новый размер файла  echo "Dir size $new\_size, $i empty files in it" #Выводим результат  done |

Результаты выполнения:



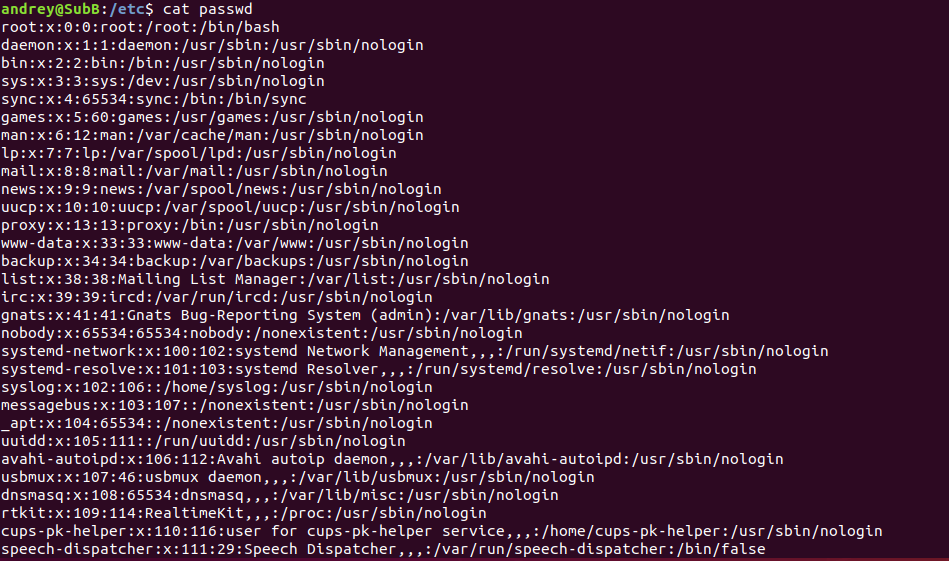
Был приведен отрывок, так как достичь максимального значения вряд ли бы удалось, так как максимальное количество файлов/каталогов зависит от inode, количество которых варьируется, но максимальное равно 2^32-1.

## 2.8. Содержимое /etc/passwd, /etc/shadow, утилита /usr/bin/passwd

Ознакомиться с содержимым /etc/passwd, /etc/shadow, с утилитой /usr/bin/passwd, проанализировать права доступа к этим файлам.

### 2.8.1. /etc/passwd

Ознакомимся с содержимым файла /etc/passwd с помощью команды cat:



Файл /etc/passwd содержит строки следующего вида:

login:password:UID:GID:GECOS:home:shell, где:

• login - имя пользователя (до 8 символов, регистр важен)

• password - хеш пароля (пароли хранятся в файле /etc/shadow)

• UID - уникальный идентификатор пользователя в пределах системы

• GID - уникальный идентификатор группы в пределах системы, к которой принадлежит пользователь

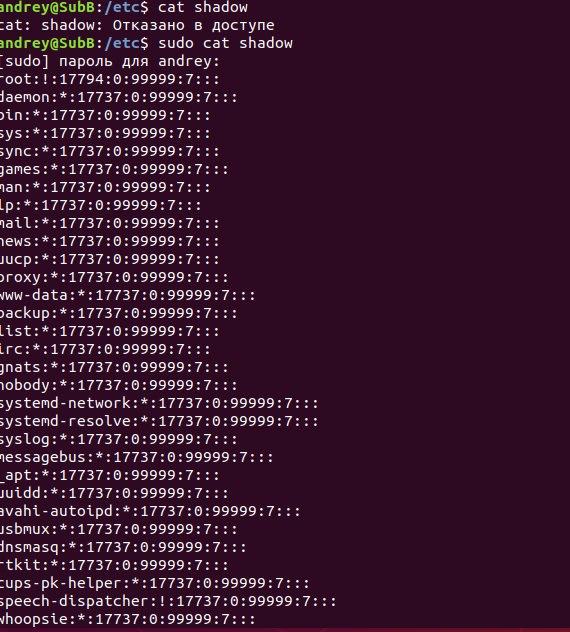
• GECOS - комментарий, расширенное описание пользователя, например, ФИО (полное имя пользователя)

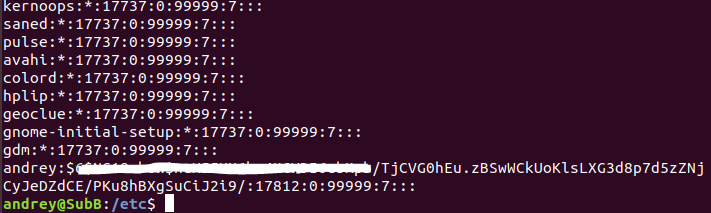
• home - домашний каталог пользователя (там хранятся все личные файлы пользователя и его web-страницы)

• shell - начальная оболочка (пользовательская)

### 2.8.2. /etc/shadow

Файл /etc/shadow, в отличие от файла /etc/passwd, доступен для чтения только суперпользователю и содержит защифрованную информацию о паролях:





Каждая строка определяет информацию о пароле конкретного аккаунта, поля в ней разделены знаком ":".

login:password:date1:days1:days2:days3:days4:date2:booking, где:

• login - имя пользователя

• password - хеш пароля

• date1 - дата последнего изменения пароля

• days1 - через сколько дней можно будет поменять пароль

• days2 - через сколько дней пароль устареет

• days3 - за сколько дней до того, как пароль устареет, начать напоминать о необходимости смены пароля

• days4 - через сколько дней после того, как пароль устареет, заблокировать учётную запись пользователя

• date2 - дата, при достижении которой учётная запись блокируется

• booking - зарезервированное поле

### 2.8.3. /usr/bin/passwd

Пользователь из группы администраторов может сменить свой пароль при помощи программы /usr/bin/passwd. После запуска программы, система предложит ввести старый пароль, затем дважды ввести новый, после этого пароль будет обновлен:

## 2.9. Права владения и доступа

Исследовать права владения и доступа, а также их сочетаемость.

### 2.9.1. Утилиты chmod, chown

Привести примеры применения утилит chmod, chown к специально созданному для этих целей отдельному каталогу с файлами.

Синтаксис утилиты chmod:

chmod [options] mode[,mode] file1 [file2 ...]

Утилита chmod предназначена для изменения прав доступа к файлам и директориям.

Флаги:

* -R рекурсивное изменение прав доступа для каталогов и их содержимого
* -f не выдавать сообщения об ошибке для файлов, чьи права не могут быть изменены.
* -v подробно описывать действие или отсутствие действия для каждого *файла*.

Права можно записывать как в восьмеричном представлении так и в символьном. В восьмеричном представлении, для стандартных прав, указываются 3 восьмеричные цифры (1-я для владельца, 2-я для группы, 3-я для всех остальных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OCT** | **BIN** | **Mask** | **Права на файл** | **Права на каталог** |
| 0 | 0 | - - - | отсутствие прав | отсутствие прав |
| 1 | 1 | - - x | права на выполнение | [доступ к файлам и их атрибутам1)](http://help.ubuntu.ru/wiki/%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B0%D0%B2%D0%B0_unix" \l "fn__1) |
| 2 | 10 | - w - | права на запись | отсутствие прав |
| 3 | 11 | - w x | права на запись и выполнение | [все, кроме доступа к именам файлов2)](http://help.ubuntu.ru/wiki/%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B0%D0%B2%D0%B0_unix" \l "fn__2) |
| 4 | 100 | r - - | права на чтение | только чтение имен файлов |
| 5 | 101 | r - x | права на чтение и выполнение | [чтение имен файлов и доступ файлам и их атрибутам3)](http://help.ubuntu.ru/wiki/%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B0%D0%B2%D0%B0_unix" \l "fn__3) |
| 6 | 110 | r w - | права на чтение и запись | только чтение имен файлов |
| 7 | 111 | r w x | полные права | все права |

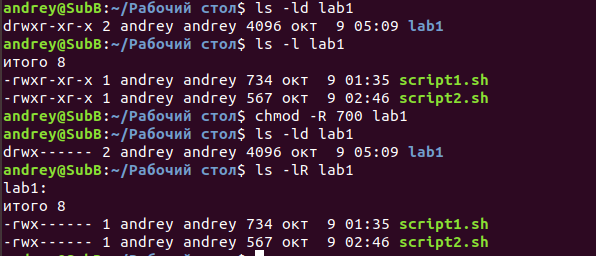
Другой способ назначения прав - это использование маски режима доступа (символьное представление). Помимо прав задается еще кому мы собираемся эти права выставлять:

* **u** - владельцу объекта;
* **g** - группе объекта;
* **o** - пользователю «все остальные»;
* **a** - все вышеперечисленное.

Для назначения прав используются три знака: минус, плюс или равно:

* **-** - убрать указанные права с объекта;
* **+** - добавить указанные права к существующим правам объекта;
* **=** - заменить права объекта на указанные.

Пример использования утилиты chmod:



Утилита chown предназначена для изменения владельца и/или группы для указанного файла.

Синтаксис:  
chown [-cfhvR] [--dereference] [--reference=rfile] пользователь[:группа] файл…

Флаги:

* -c, --changes

Подробно описывать действие для каждого файла, владелец которого действительно изменяется.

* -f, --silent, --quiet

Не выдавать сообщения об ошибках для файлов, чей владелец не может быть изменён.

* -h, --no-dereference

Работать с самими символьными ссылками, а не с файлами, на которые они указывают. Данная опция доступна только если имеется системный вызов lchown.

* -R, --recursive

Рекурсивное изменение владельца каталогов и их содержимого.

* -v, --verbose

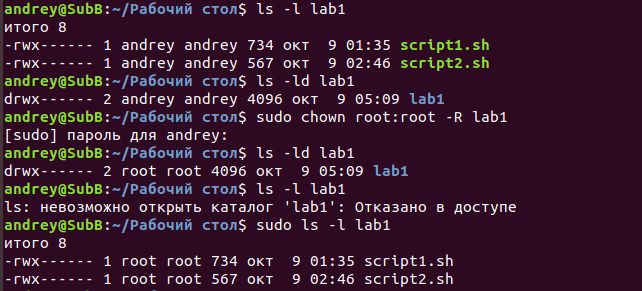
Подробное описание действия (или отсутствия действия) для каждого файла.

* --dereference

Изменить владельца файла, на который указывает символьная ссылка, вместо самой символьной ссылки.

* --reference=rfile

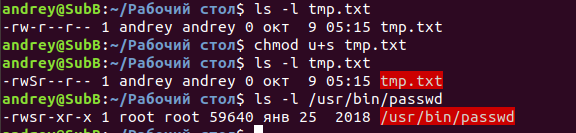
Изменить владельца файла на того, который является владельцем файла.



### 2.9.2. Флаг SUID

Расширить права исполнения экспериментального файла с помощью флага SUID.

Обычно в правах доступа присутствуют символы rwx, однако существует символ s. Данный флаг необходим чтобы пользователь смог запустить файл, который ему не принадлежит.

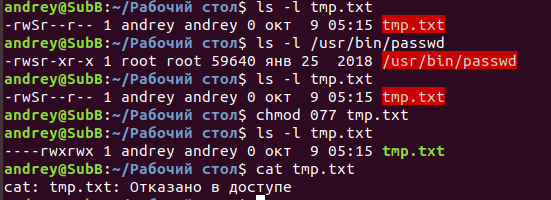


Как видно, наличие флага SUID отображается в списке прав доступа при помощи буквы s вместо x. Если права на выполнение у файла нет, то буква S пишется в верхнем регистре.  
Как пример, утилита /usr/bin/passwd имеет флаг «s».

### 2.9.3. Процесс формирования прав

Экспериментально установить, как формируются итоговые права на использование файла, если права пользователя и группы, в которую он входит, различны.

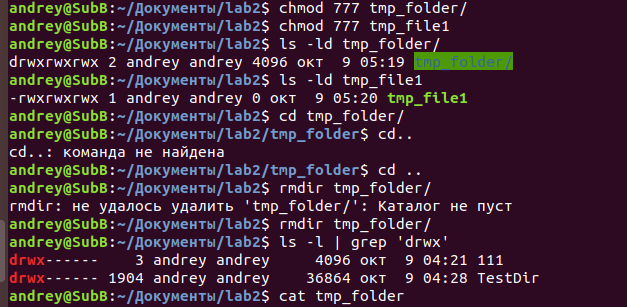
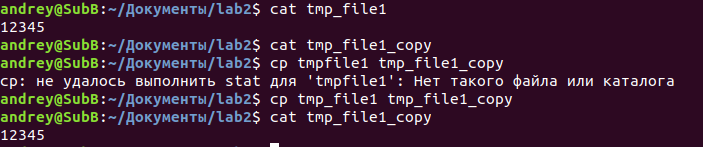
Обнулим права пользователя, и предоставим все права группе.



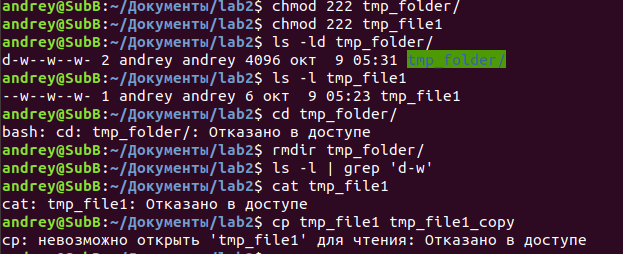
Не смотря на то, что пользователь принадлежит к группе, отсутствие у него прав не позволяют получить доступ к файлу. Вывод: приоритет пользовательских прав выше прав группы.

### 2.9.4. Варьирование прав доступа

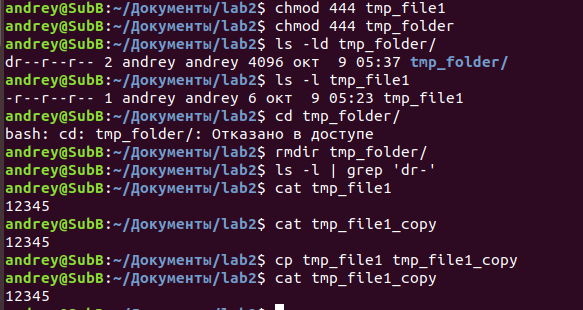
Сопоставить возможности исполнения наиболее часто используемых операций, варьируя правами доступа к файлу и каталогу.

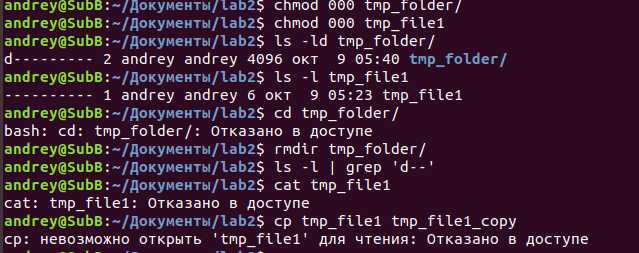
Проведем ряд тестов над файлом и каталогом, используя следующий операции: ls, cd, rmdir, cat, cp. Для тестов будем использовать специально созданный каталог tmp\_folder и файл tmp\_file1.

В первом тесте были даны все возможные права.

Во втором тесте были даны права лишь на запись.

В третьем тесте были даны права лишь на чтение.



В четвертом тесте были убраны все права.

## 2.10. Программа-шлюз

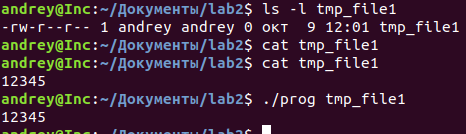
Разработать «программу-шлюз» для доступа к файлу другого пользователя при отсутствии прав на чтение информации из этого файла. Провести эксперименты для случаев, когда пользователи принадлежат одной и разным группам. Сравнить результаты. Для выполнения задания применить подход, аналогичный для обеспечения функционирования утилиты /usr/bin/passwd (манипуляции с правами доступа, флагом SUID, а также размещением файлов).

В качестве программы-шлюза была написана программа на С++ lab2\_9.cc:

|  |
| --- |
| #include <stdio.h>  #include <fstream>  #include <iostream>  using namespace std;  int main(int argc, char \*\* argv){  if(argc<2){ // Если количество аргументов меньше 2  printf("ERROR. No input file!\n"); // Вывод сообщения об ошибке  return 1; // Завершение программы  }  ifstream fin(argv[1]); // Открываем на чтение файл который ввел пользователь  if(!fin.is\_open()){ // Если файл не открыт  printf("ERROR. Cannot open file"); // Вывод сообщения об ошибке  return 1; // Завершение программы  }  char temp; // Переменная, в которую будут читаться символы  while(fin>>temp) // Пока в файле есть символы на чтение  printf("%c",temp); // Вывод символов в консоль  printf("\n"); // Перевод строки  fin.close(); // Закрытие потока чтения файла  return 0; // Завершение программы  } |

Первоначально флаг SUID не установлен. Результаты при работе от владельца:

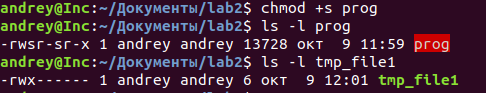
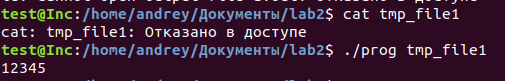




Результаты при работе от другого пользователя:



Установим флаг SUID. Результаты при работе от произвольного пользователя:

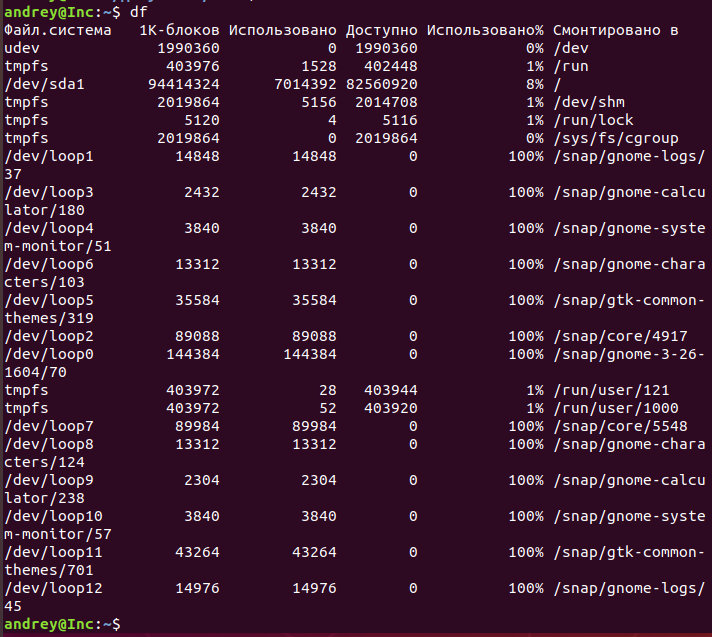
В консоль вывелось содержимое файла.

## 2.11.Утилиты df и аналогичные

Применяя утилиту df и аналогичные ей по функциональности утилиты, а также информационные файлы типа fstab, получить информацию о файловых системах, возможных для монтирования, а также установленных на компьютере реально.

Привести информацию об исследованных утилитах и информационных файлах с анализом их содержимого и форматов.

Утилита df предоставляет информацию о состоянии жесткого диска и точках монтирования.

Как видно, на компьютере 8 точек монтирования.

• udev – менеджер устройств в ядре Linux.

• tmpfs – одна из разновидностей файловых систем, отличающаяся быстрой скоростью работы и надежностью за счет того что располагается в оперативной памяти

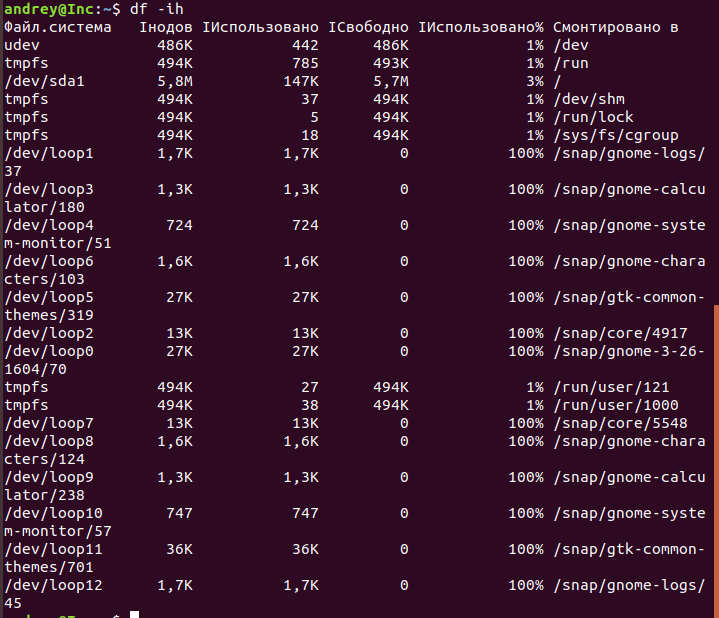
• /dev/sda1 - это основной раздел, который отформатирован под Linux

Для каждой файловой системы указан размер, кол-во использованного и доступного пространства. Справа указана точка монтирования. То есть указание, в какую директорию нужно перейти, чтобы оказаться в выбранной файловой системе.

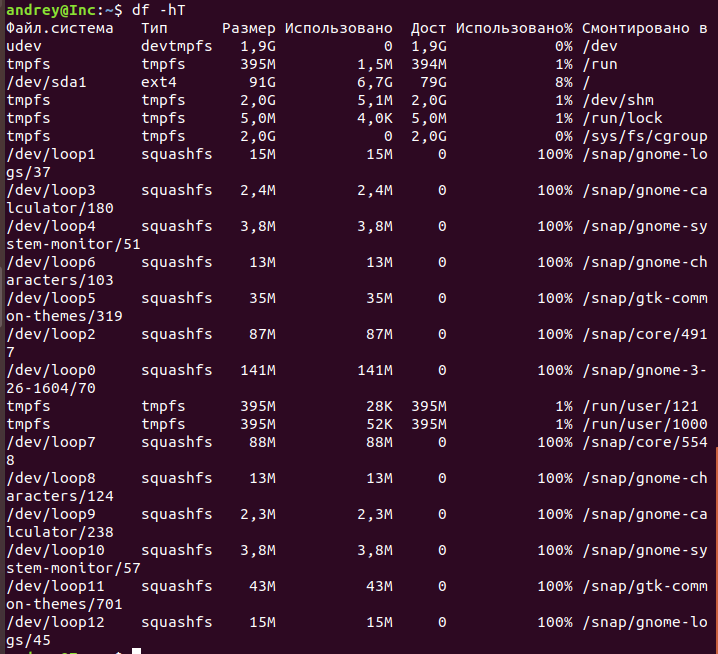
Основные флаги утилиты df:

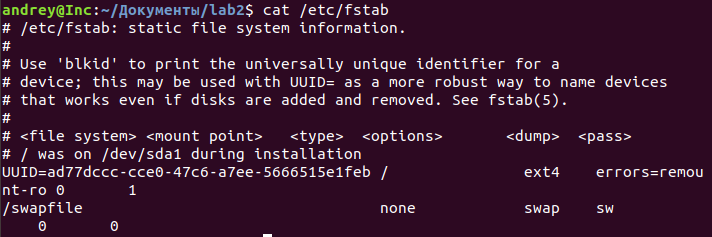
* -a - полный вывод систем
* -h - показ результата в степенях 1024
* -i - показ размеров в виде инодов взамен блоков
* -T - вывод типа файловой системы
* -x - не показывать данную систему

Показ размера в inode’ах:



Фильтр отображаемой системы:

Содержимое файла /etc/fstab, который содержит информацию о различных файловых системах и устройствах хранения информации компьютера:

Каждая запись имеет следующие поля (которые разделяются пробелом или табуляцией):  
<file system> <dir/mount point> <type> <options> <dump> <pass>

• Поле <file system> сообщает, что монтировать, имя монтируемого устройства.

• Поле <dir dir/mount point > определяет путь, по которому будет смонтирована файловая система.

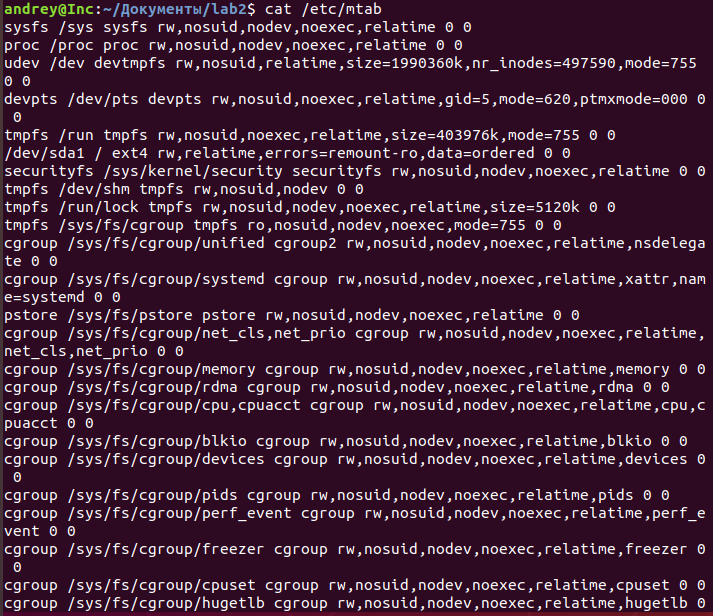
• Поле <type> определяет тип ФС. В данном случае это ext4.

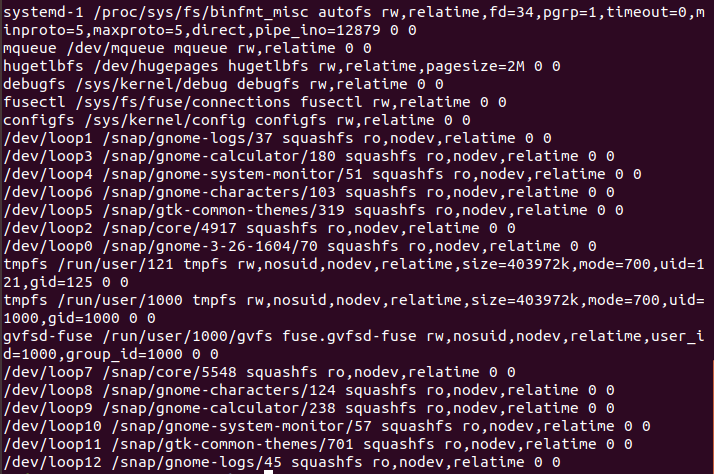
• Поле <options> содержит список возможных опция для ФС. Таких как автоматическое монтирование, разрешение исполнения бинарных файлов, разрешение или запрет монтирования системы для пользователя и так далее.

• Поле <dump> отвечает за резервную копию. Если при подключении ФС dump установлен в 0, то система игнорируется. Если 1, создается резервная копия.

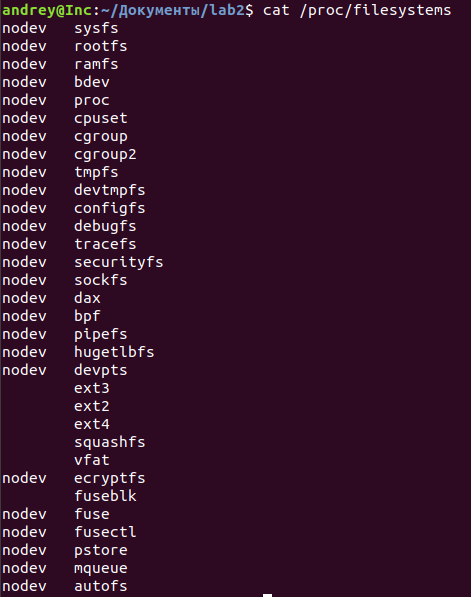
• Поле <pass> отвечает за приоритет при проверке. У корневой ФС pass равен 1. У остальных систем должен быть установлен в 2.

В файле mtab прописаны устройства, смонтированые в систему в настоящий момент. Когда программа mount подключает файловую систему, она дописывает соответствующую строку в /etc/mtab. Когда umount отключает файловую систему, из этого файла соответствующая строка удаляется. Формат файла аналогичен формату файла /etc/fstab.

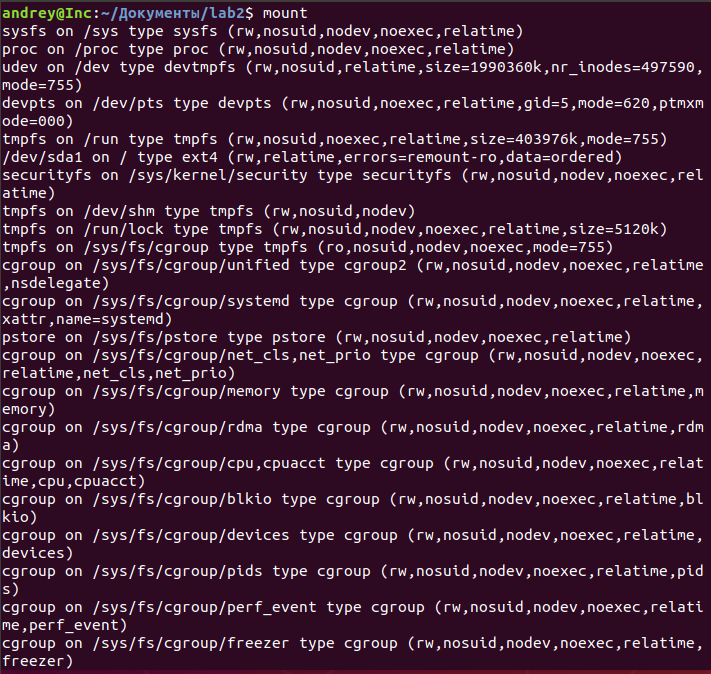


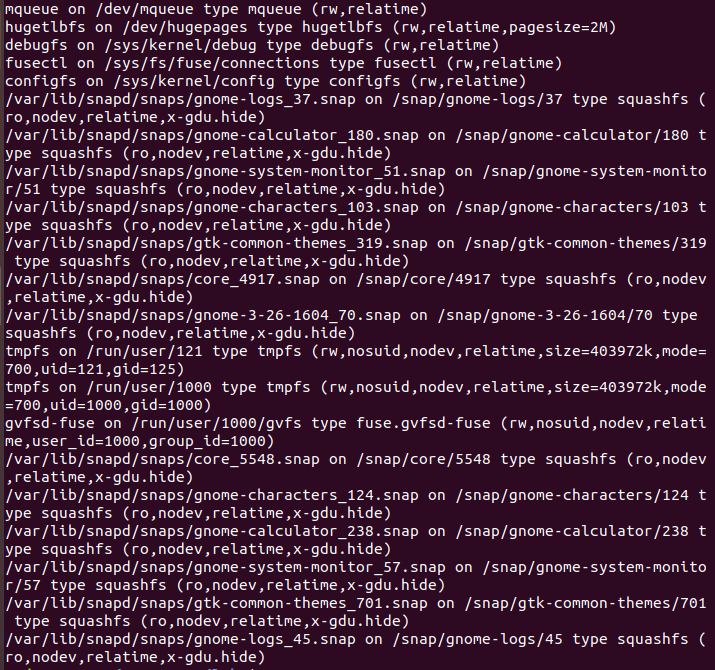


Список всех систем, доступных для монтирования находится в файле cat /proc/filesystems. Первая строка “nodev” показывает, что системе не требуется блочное устройство для установки, это виртуальная файловая система.

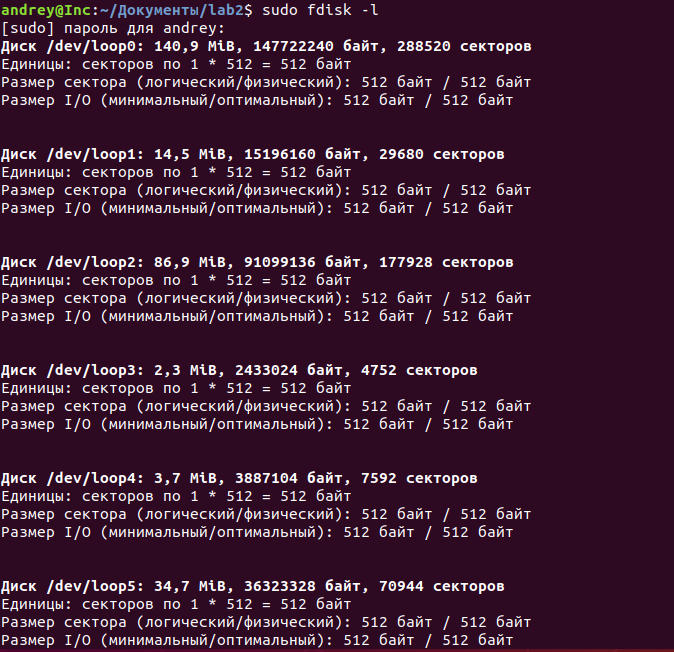


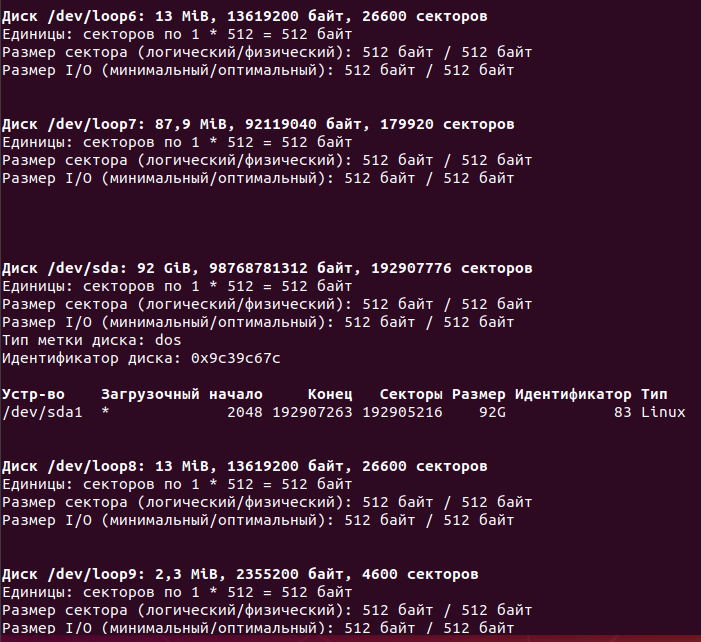
Команда mount выводит точки монтирования всех устройств и файловых систем. Формат строки таблицы: имя устройства, режим включения(on), точка монтирования, тип файловой системы.

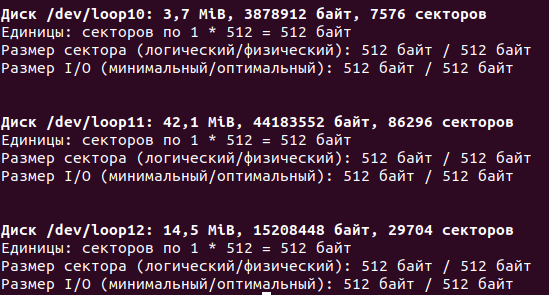


В Linux подключение раздела в основную ФС называется монтированием. Точка монтирования – это место, к которому надо подключить раздел.

Утилита fdisk позволяет просмотреть информацию о дисках и разделах этих дисков и промежуток занимаемых разделом секторов, размер блока для каждого диска.







В Linux максимальная длина пути определена в файле /usr/include/linux/limits.h. По умолчанию она равна 4096, включая нулевой символ в конце строки.

## 

## 2.12. Утилита file

Проанализировать и пояснить принцип работы утилиты file

### 2.12.1. Алгоритм функционирования утилиты file

Привести алгоритм её функционирования на основе информационной базы, размещение и полное имя которой указывается в описании утилиты в технической документации ОС (как правило, /usr/share/file/magic.\*), а также содержимого заголовка файла, к которому применяется утилита. Определить, где находятся магические числа и иные характеристики, идентифицирующие тип файла, применительно к исполняемым файлам, а также файлам других типов.

Утилита идентификации файлов file выполняет ряд проверок для указанного файла, пытаясь класcифицировать его. Сначала происходит тест на файловую систему, затем на магические цифры и затем языковые тесты. Первый пройденный тест прерывает проверку и возвращает результат. Если файл является файлом-директорией текстовым, файлом исходного кода, скомпилированным файлом и так далее, то утилита сообщит об этом.

Проверка на файловую систему – это проверка, не является ли файл сокетом, символьной ссылкой, или FIFO.

Магические числа – проверка на файлы в определённых форматах (.out, .exe и прочие). Они описаны в usr/share/file/magic.mgc – именно туда обращается утилита file.

При попытке вытащить содержимое файла magic.mgc мы получим набор символов с неопределённой кодировкой (рез-т получен с помощью cat magic.mgc; это лишь малая часть):

### 

### 2.12.2. Утилита file с разными ключами

Утилиту file выполнить с разными ключами.

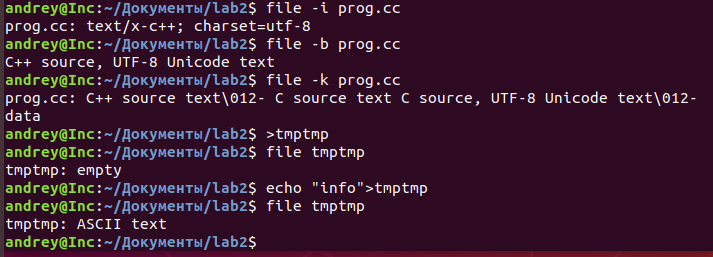
// без ключей

### 

// i – выводим в mime-подобном стиле (менее удобен для восприятия человеком)

// b – не выводить имя проверяемого файла

// k – при нахождении совпадения не останваливаться, а проверять далее



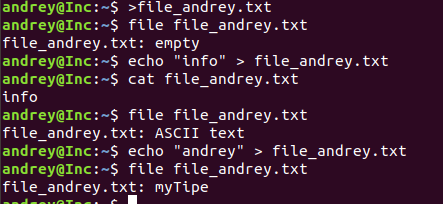
### 2.12.3. Добавление собственного типа файла

Привести экспериментальную попытку с добавлением в базу собственного типа файла и его дальнейшей идентификацией. Описать эксперимент и привести последовательность действий для расширения функциональности утилиты file и возможности встраивания дополнительного типа файла в ФС (согласовать содержимое информационной базы и заголовка файла нового типа).

Добавим в etc/magic собственный тип файла с его идентификацией:

Запись имеет следующий смысл:

* 1. Начинать поиск с 0 бита
  2. Искать строку...
  3. ...значение которой должно быть andrey
  4. Имя нового типа – myType



# 3. Вывод

В ОС Linux существует несколько типов файлов: обычные, директории, ссылки, сокеты, очереди, блок-ориентированные файлы, байт-ориентированные файлы.

Благодаря использовнию ссылок, у одного файла может быть несколько путей, т.е. несколько файлов в структуре каталогов Linux могут быть физически одним файлом на диске. Это достигается тем, что в файловой системе каждый файл идентифицируется уникальным номером, называемым inode (Индексный дескриптор). Индексные дескрипторы хранят информацию о файлах, такую как принадлежность владельцу (пользователю и группе), режим доступа (чтение, запись, запуск на выполнение) и тип файла.

Были разобраны специальные файлы /etc/passwd, /etc/shadow, в которых хранится информация об именах пользователей, пароли в зашифрованном виде, идентификаторы пользователей, идентификаторы групп пользователей, а так же пути к домашним каталогам пользователей.

Кроме того, было установлено, что, используя программу-шлюз, мы можем получить доступ к файлам, к которым у нас нет права доступа.

Также было произведено знакомство с утилитами:

• file – для определения типа файла

• df/lsblk/sfdisk – инструментами, позволяющими получить информацию о используемых дисках

• od – для проведения анализа содержимого файла