Лабораторная работа №3
«Работа с текстовой информацией в Astra Linux»

1. Теоретическая часть

1.1. Введение

На заре вычислительной техники первые компьютеры (мэйнфреймы) не содержали графических оболочек, поэтому все операции проводились при помощи терминала, а вся информация представлялась исключительно в виде текста. Нужны были программы, которые помогали бы операторам находить нужные данные, перерабатывать их, редактировать и т.д.

В настоящей лабораторной работе мы сделаем обзор утилит, которые предназначены для обработки текстовой информации. Они появились ещё во времена операционной системы UNIX в 70-е года, однако до сих пор не потеряли своей актуальности и активно используются системными администраторами в современных дистрибутивах GNU/Linux благодаря совместимости со стандартом POSIX.

Одна из причин, которые делают командную оболочку бесценным инструментом сегодня, — это большое количество команд обработки текста и возможность легко объединять их в конвейер, создавая сложные шаблоны обработки. Эти команды делают тривиальными многие задачи по анализу текста и данных, преобразованию данных между разными форматами, по фильтрации строк и т. д.

При работе с текстовыми данными главный принцип заключается в том, чтобы разбить любую сложную проблему на множество более мелких — и решить каждую из них с помощью специализированного инструмента. Именно в этом состоит суть философского подхода к разработке: «пишите программы, которые делают что-то одно и делают это хорошо».

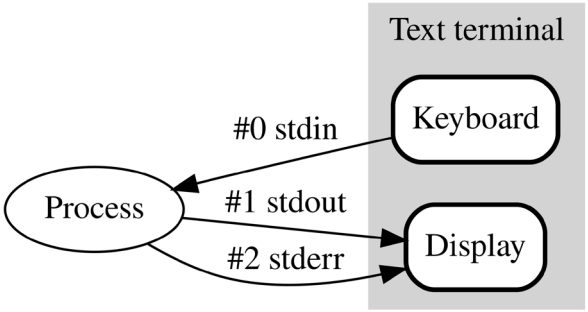
1.2. Стандартные потоки

Одним из нескольких новаторских достижений UNIX были абстрактные устройства, которые устранили необходимость в том, чтобы программа знала или заботилась о том, с какими устройствами она взаимодействует. Тогда была введена концепция потока данных — упорядоченной последовательности байтов данных.

Стандартный ввод (stdin) — это поток, из которого программа считывает свои входные данные. Программа запрашивает передачу данных с помощью операции чтения.

Не всем программам требуется потоковый ввод. Например, программа ***ls*** может принимать аргументы командной строки, но при этом выполнять свои операции без ввода каких-либо потоковых данных.

Стандартный вывод (stdout) — это поток, в который программа записывает свои выходные данные. Программа запрашивает передачу данных с помощью операции write.



Не все программы генерируют выходные данные. Например, команда ***mv*** не сообщает о результатах своей работы.

Стандартный поток ошибок (stderr) - это другой выходной поток, обычно используемый программами для вывода сообщений об ошибках или диагностики.

1.3. Перенаправление потоков ввода и вывода

Вывод данных на экран и чтение их с клавиатуры происходит потому, что по умолчанию стандартные потоки ассоциированы с терминалом пользователя. Это не является обязательным — потоки можно подключать к чему угодно — к файлам, программам и даже устройствам. В командном интерпретаторе ***bash*** такая операция называется перенаправлением.

Используются следующие символы перенаправления потока:

Символ	Комментарий
< file.txt	Использовать файл **file.txt** как источник данных для стандартного потока ввода в программу.
> file.txt	Направить стандартный поток вывода в файл **file.txt**. Если файл не существует, он будет создан; если существует — перезаписан сверху.
2> file.txt	Направить стандартный поток ошибок в файл **file.txt**. Если файл не существует, он будет создан; если существует — перезаписан сверху.
>> file.txt	Направить стандартный поток вывода в файл **file.txt**. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.
2>> file.txt	Направить стандартный поток ошибок в файл **file.txt**. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.
&> file.txt или **>& file.txt**	Направить **stdout** и **stderr** в файл **file.txt**. Другая форма записи: **> file.txt 2>&1**.

Рассмотрим примеры:

Вывод информации о текущем времени в файл date.txt:

```
timedatectl > date.txt
```

Добавление информации о текущем Unix-времени в файл date.txt:

```
date +%s >> date.txt
```

1.4. Объединение команд

Когда команда выполняется, результат её выполнения записывается в особую переменную — «**?**». Результат выполнения может быть:

- Успешным. Тогда в переменную «**?**» запишется «0»;
- Не успешным, то есть при выполнении были ошибки. Тогда в переменную «**?**» запишется не ноль, а какое-нибудь другое число. Обычно это число зависит от ошибки.

Посмотреть значение этой переменной можно с помощью команды

```
echo $?
```

Например, при введении команды ***ls*** будет выведено содержимое текущей директории. Тогда в переменной **?** будет лежать 0. А при введении команды

```
ls /test
```

будет сообщено об ошибке, а в переменной **?** будет лежать число 2.

Существует два метода объединения команд: с помощью ***логических операций*** и ***каналов***. Логические операции позволяют объединять команды в некоторые связки для выполнения их по очереди.

Символ	Комментарий
A ; B	Оператор “;” выполняет все команды независимо от того, завершились ли предыдущие неудачно или нет
A && B	Оператор “&&” или И выполняет вторую команду только в том случае, если предыдущая команда выполнена успешно
A || B	Оператор “||” ИЛИ выполняет вторую команду только в том случае, если предыдущая команда возвращает ошибку.

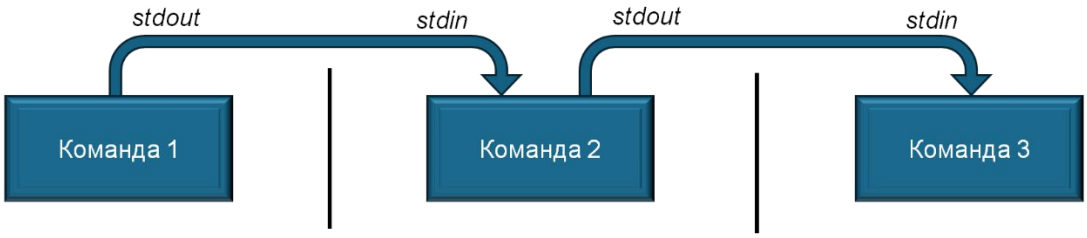
Например, мы можем прочитать какой-либо файл и после вывести строку в консоль. При этом следующая команда может зависеть от результата предыдущей команды.

```
cat file ; echo I will always work
cat file && echo I will work only when file exists
cat file || echo I will work only when file NOT exists
```

Pipes (каналы) используются для перенаправления потока из одной программы в другую.

Они нужны для перенаправления **stdout** одного процесса на **stdin** другого. Именно вывод утилиты, а не содержимое файла.

С помощью символа «|» можно вызвать несколько команд одновременно, после чего вывод одной команды передаётся на вход другой по конвейерному типу, как представлено на рисунке ниже:



Например,

```
 timedatectl | grep “Local time”
lsb_release -a  > test.txt; ls >> test.txt; cat test.txt | grep a
```

1.5. Утилита grep

Название команды ***grep*** расшифровывается как "search **g**lobally for lines matching the **r**egular **e**xpression, and **p**rint them". Это одна из самых востребованных команд в терминале Linux, которая входит в состав проекта GNU. До того как появился проект GNU, существовала утилита предшественник ***grep***, с тем же названием, которая была разработана в 1973 году Кеном Томпсоном для поиска файлов по содержимому в Unix. А потом уже была разработана свободная утилита с той же функциональностью в рамках GNU.

grep дает очень много возможностей для фильтрации текста. Вы можете выбирать нужные строки из текстовых файлов, отфильтровать вывод команд, и

даже искать файлы в файловой системе, которые содержат определённые строки. Утилита имеет следующий синтаксис:

```
grep [ключи] шаблон [имя_файла ...]
```

Для получения нужных результатов используются следующие ключи:

Ключ	Описание
-c	Выдает только количество строк, содержащих выражение.
-h	Скрывает вывод названия файла, в котором было обнаружено вхождение. Используется при поиске по нескольким файлам.
-i	Игнорирует регистр символов при поиске.
-l	Выдает только имена файлов, содержащих сопоставившиеся строки.
-n	Выдает перед каждой строкой ее номер в файле (строки нумеруются с 1).
-s	Скрывает выдачу сообщений о не существующих или недоступных для чтения файлах.
-v	Выдает все строки, за исключением содержащих выражение.
-E	Поиск с использованием регулярных выражений.
-o	Вывод только обнаруженных символов.
-r	Рекурсивный поиск.

Рассмотрим следующие примеры:

Найдём строку Hello в файле file.txt:

```
grep Hello file.txt
```

Найдём строку Hello во всех файлах текущей директории:

```
grep -r Hello
```

Найдём число вхождений строки Hello в файле file.txt:

```
grep -c Hello file.txt
```

Регулярные выражения (regexp или regex) — это механизм для поиска и замены текста по шаблону.

Выражения используются в многих программах, не только в связке с ***grep***.

Существуют следующие метасимволы, с помощью которых можно производить поиск:

Метасимвол	Описание работы
\	Начало буквенного спецсимвола
^	Указывает на начало строки
$	Указывает на конец строки
*	Указывает, что предыдущий символ может повторяться 0 или больше раз
+	Указывает, что предыдущий символ должен повторится больше один или больше раз
?	Предыдущий символ может встречаться ноль или один раз
{n}	Указывает сколько раз (n) нужно повторить предыдущий символ
{N,n}	Предыдущий символ может повторяться от N до n раз
.	Любой символ кроме перевода строки
[az]	Любой символ, указанный в скобках
x|y	Символ x или символ y
[^az]	Любой символ, кроме тех, что указаны в скобках
[a-z]	Любой символ из указанного диапазона
[^a-z]	Любой символ, которого нет в диапазоне
[[:alpha:]]	Является алфавитным символом
[[:digit:]]	Является числом

Рассмотрим следующие примеры:

Поиск содержимого файлов, начинающихся с символов A или B:

```
grep –re ‘^[AB]’
```

Поиск количества строк в каждом файле, содержащие подряд две буквы «в»:

```
grep –rc “[в]{2}”
```

Поиск содержимого файлов, содержащих слова «Вы или вы»:

```
grep –re ‘[Вв]ы’
```

Поиск в файле IPv4 адресов (для любителей реальной практики):

```
grep –E ‘[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}’ ip.txt
```

1.6. Утилита find

Очень важно уметь вовремя найти нужную информацию в системе. Конечно, все современные файловые менеджеры предлагают отличные функции поиска, но им не сравнится с поиском в терминале Linux. Он намного эффективнее и гибче обычного поиска, вы можете искать файлы не только по имени, но и по дате добавления, содержимому, а также использовать для поиска регулярные выражения. Кроме того, с найденными файлами можно сразу же выполнять необходимые действия.

find — утилита, с помощью которой возможно найти файл по его имени. Она имеет следующий синтаксис:

```
find directory-to-search criteria action
```

При этом:

- **directory-to-search** (каталог поиска) — это отправной каталог, с которой ***find*** начинает поиск файлов по всем подкаталогам, которые находятся внутри. Если не указать путь, то поиск начнется в текущем каталоге;
- **criteria** — критерии, по которым нужно искать файлы;
- **action** (действие) — указатель того, что сделать с каждым найденным файлом, соответствующего критериям.

Приведём описание критериев в следующей таблице:

<table>
<tr><th>Критерий</th><th>Описание</th><th colspan="2">Комментарии</th></tr>
<tr><td>-name</td><td>Поиск по имени файла</td><td colspan="2">Учитывается регистр. Критерий -iname не учитывает регистр.</td></tr>
<tr><td>-size</td><td>Поиск по размеру файла</td><td colspan="2">"+": Поиск файлов больше заданного размера;
"-" : Поиск файлов меньше заданного размера;
Отсутствие знака означает, что размер файлов в поиске должен полностью совпадать.</td></tr>
<tr><td>-type</td><td>Поиск по типу файла</td><td colspan="2">f – простые файлы;
d – каталоги;
l – символические ссылки;
b – блочные устройства (dev);
c – символьные устройства (dev);
p – именованные каналы;
s – сокеты.</td></tr>
<tr><td>-empty</td><td>Поиск пустых каталогов и файлов</td><td colspan="2"></td></tr>
<tr><td>-Xmin</td><td rowspan="2">Поиск по времени</td><td rowspan="2">X – a (access, поиск по времени доступа);
X – m (modify, поиск по времени изменения);
X – c (change, поиск по времени изменения метаданных);</td><td>-T – поиск файлов раньше T (мин)
+T – поиск файлов позже T (мин)</td></tr>
<tr><td>-Xtime</td><td>-T – поиск файлов раньше T (дней)
+T – поиск файлов позже T (дней)</td></tr>
</table>

После нахождения файлов можно произвести над ними следующие действия:

Действия	Описание
-delete	Удаление всех результатов поиска
-exec	Выполнение команды в каждой найденной строке

Рассмотрим следующие примеры:

Найти файл с названием MPSU:

```
find . -name MPSU
```

Вывести все файлы с информацией о них:

```
find . -printf '%M %n %s %Tb %p\n'
```

Вывести все файлы, размером более 100 МБ, но менее 2ГБ:

```
find . -size+100M -size -2G
```

Вывести все файлы, изменённых не ранее 30 и не позднее 20 минут назад:

```
find . -type f -mmin +20 -mmin -30
```

1.7. Утилита cut

Достаточно часто нам нужно взять из набора текста только несколько слов или вырезать и вывести на экран определённые символы.

Утилита ***cut*** позволяет вырезать символы из каждой переданной ей строки. Она имеет следующий синтаксис:

```
cut ключ путь_к_файлу
```

Часто используются следующие ключи:

Ключ	Описание
-c	Вырезать указанную последовательность символов
-d	Указать символ разделения (по умолчанию TAB)
-f	Выбрать символы, разделенные определенным символом
-s	Указание на пропуск любой строки, в которой нет разделителя

При этом мы можем задать форматы для задания списка полей или колонок:

Формат	Описание
A-B	Поля или колонки от A до B включительно
A-	От поля или колонки A до конца строки
-B	С начала строки до поля или колонки B
A, B	Поля или колонки A и B

Рассмотрим следующие примеры:

Вывести первые пять символов файла:

```
cut -c 1-5 1.txt
```

Выделить второй столбец из строки, символ разделения - двоеточие:

```
cut -d ':' -f 2 1.txt
```

Выделить первое слово из текста (символ разделения - пробел), если их нет в строке – строка пропускается

```
cut -d' ' -f1 -s 1.txt
```

1.8. Утилита tr

Утилита ***tr (translate)*** — утилита командной строки, позволяющая посимвольно обрабатывать текст.

Она позволяет избавиться от повторяющихся пробелов, изменить регистр текста или зашифровать его с помощью посимвольного сдвига. Имеет следующий синтаксис:

```
tr [ключи]... набор1 [набор2]
```

Укажем полезные на практике ключи:

Ключ	Описание
-c	Оставить только символы из первого набора, остальные заменить на значение из второго набора
-d	Удалить символы, входящие в первый набор
-s	Заменяет все символы из первого набора, встречающиеся несколько раз подряд на одиночное вхождение

Рассмотрим следующие примеры:

Заменить все строчные символы на прописные:

```
tr "a-z" "A-Z" < 1.txt
tr "[[:lower:]]" "[[:upper:]]" < 1.txt
```

Заменить все символы, кроме a-z, \n на символ X:

```
tr -c "a-z\n" X < 1.txt
```

Удалить все строчные символы:

```
tr -d "\n" < 1.txt
```

Заменить все повторяющиеся пробелы на один:

```
tr -s " " < 1.txt
```

1.9. Утилита wc

wc (word count) – утилита командной строки, выводящая число переводов строк, слов и байт для каждого указанного файла и итоговую строку, если было задано несколько файлов.

Стоит обратить внимание, что размер байтов в файле — то, сколько реальной памяти он занимает, а вот количество символов — количество печатных, читаемых символов, которые можно декодировать в текст. Имеет следующий синтаксис:

```
wc [ключи]... [ имя_файла ... ]
```

Используемые ключи:

Ключ	Описание
-c	Вывести размер файла в байтах
-m	Вывести количество символов в файле
-l	Вывести количество строк в файле
-w	Вывести количество слов в файле

Вызов без параметров вернёт все значения количества строк, слов и символов в файле. Пример:

Вывести число строк в файле:

```
wc -l file.txt
```

1.10. Утилита uniq

uniq — утилита командной строки, которая предназначена для поиска одинаковых строк в массивах текста.

Имеет следующий синтаксис

```
uniq [OPTION]... [INPUT [OUTPUT]]
```

Используемые ключи:

Ключ	Описание
-u	Вывести исключительно те строки, у которых нет повторов
-d	Вывести все строки, удаляя дубликаты
-D	Вывести только повторяющиеся строки
-c	В начале каждой строки вывести число, которое обозначает количество повторов

Рассмотрим пример:

Вывести число уникальных строк в файле:

```
uniq -u file.txt
```

1.11. Утилита sort

sort — утилита для вывода текстовых строк в определенном порядке.

Имеет следующий синтаксис:

```
sort [OPTION]... [FILE]
```

Используемые ключи:

Ключ	Описание
-r	Сортировка в обратном порядке
-u	Игнорировать повторяющееся строки (удаление дубликатов)
-kχ	Сортировка по колонке χ

Пример:

Сортировать текст файла по 8 столбцу в обратном порядке:

```
sort –r –k8 file.txt
```

1.12. Утилита sed

sed (stream editor) — потоковый редактор текста.

На вход получает данные построчно и редактирует каждую строку в соответствии с правилами. Сделан по образцу строкового редактора ***ed***, с помощью которого можно было командами нацеливаться на определённую позицию и редактировать текст. Имеет следующий синтаксис:

```
sed [параметры]'правила_отбора инструкции' [файлы]
```

Рассмотрим следующие правила:

Правило	Описание	
a	Добавить строки	
d	Nd	Удалить выбранные строки по их номеру
	/text/d	Удалить строки, содержащие text
Np	Печать строк на экран. Без указания флага –n происходит дублирование обрабатываемых строк. Печатаются N-ая строка. Возможно указание нескольких строк через запятую. Без указания числа N печатаются все строки	
s/что_заменять/на_что_заменять/опции	Замена текста в строках. Опция g позволяет произвести все замены в строке. Без нее замена только первого вхождения	

Рассмотрим следующие примеры:

Вывести 1ю строку из файла:

```
sed -n ' 1p' text.txt
```

Удалить 2 строку и вывести результат на экран:

```
sed -n '1d;p' text.txt
```

Удалить из программы все пустые строки и все комментарии, начинающиеся с #:

```
sed '/^$/d; /^#/d' text.txt
```

Заменить все символы A на символ B:

```
sed 's/A/B/g' text.txt
```

Заменить строку, содержащую поле “ip=“ на значение “ip=192.168.122.2” (*ключ -i позволяет сохранять изменения в исходном файле*).

```
sed '/ip=/d;1a ip=192.168.122.2' text.txt
```

1.13. Утилита awk

awk — скриптовый язык построчного разбора и обработки входного потока данных.

По сути ***awk*** является целым языком программирования. Утилита последовательно применяется к каждой из строчек. Для фильтрации строк применяется шаблон, накладывающий ограничение на отбираемые строки. Имеет следующий синтаксис:

```
awk [ключи] '[шаблон] {действие}'
```

При этом утилита имеет расширенный вариант:

```
BEGIN {действие}                -> Выполняется до обработки строк
   шаблон {действие}
   шаблон {действие}
END {действие}                  -> Выполняется после обработки строк
```

Используемые ключи:

Ключ	Описание
-F fs	Указать символ разделения (по умолчанию пробел)
-v var=val	Задать значение переменной

Пользователь может использовать встроенные переменные или создавать свои. ***awk*** рассматривает переменную как строковую. По умолчанию строка инициализируется символом "0" или пустой строкой.

Существуют следующие типы переменных:

- позиционные;
- числа с плавающей точкой;
- строка символов;
- массив.

Утилита ***awk*** поддерживает стандартные конструкции языка C – ветвления и циклы.

Основные встроенные переменные	**Описание**
$0	Значение обрабатываемой строки
$1 - $N	Значение определенного столбца, полученных в результате разделения строки сепаратором
FS	Разделитель полей записи на вводе
NF	Число полей в текущей записи
NR	Номер записи (общее число считанных записей)
Некоторые команды	**Описание**
print	Вывести строку целиком
length(N)	Длина N в символах

Рассмотрим следующие примеры, которые производятся над файлом, содержащим значения ФИО студентов, их дату рождения и оценку за экзамен:

Напечатать весь текст:

```
awk '{print}' test.txt
```

Напечатать сообщение приветствия и прощания между текстом:

```
awk 'BEGIN {print "Hello"} {print} END {print "Good Bye"}' test.txt
```

Напечатать первый и второй столбец текста (Фамилию и имя):

```
awk '{print $1, $2}' test.txt
```

Напечатать целиком все строки, содержащие символ F:

```
awk '/F/{print $0}' test.txt
```

Напечатать второй столбец текста, символ разделитель - тире:

```
awk –F- '{print $2}' test.txt
```

Вывести число строк в файле:

```
awk 'END{print NR}' test.txt
```

Вывести сумму значений 5-го столбца:

```
awk '{sum+=$5} END{print sum}' test.txt
```

Вывести все строки, где пятый столбец равен 90:

```
awk '{if ($5==90) print $0}' test.txt
```

Вывести все строки, где пятый столбец равен 90 и вывести их количество:

```
awk '{if ($5==90) {L+=1; print $0}} END{print L}' test.txt
```

Вывести сообщение о результате аттестации в случае получения оценки:

```
awk '{if ($5>=90) {print $1 " " $2 ": Zachet"} else {print $1 " " $2 ": Peresdacha"}}'
```

Посчитать количество символов в каждом слове файла:

```
awk '{ for(i=1; i<=NF; i++) {L+=length($i)}} END {print L}' test.txt
```

Пояснение: **for(i=1; i<=NF; i++)** — стандартный цикл. При обращении к **$i** получаем значения всех столбцов по очереди с первого по последний.

1.14. Утилита xargs

xargs — утилита для передачи вывода одной команды в качестве аргументов другой. Имеет следующий синтаксис:

```
[команда_генератор_списка] | xargs [опции_xargs]
```

Ключ	Описание
-0	Использовать в качестве разделителя нулевой символ (символ окончания строки)
-d	Использовать нестандартный разделитель для строк

Примеры:

Найти все архивы и удалить их:

```
find . -name "*.tar"| xargs rm -rf
```

Найти все текстовые файлы и добавить их в архив:

```
find .-name "*.txt" -type f -print0 | xargs -0 tar -cf
```

2. Практическая часть

2.1. Задание 1

*Примечание: используйте утилиту **find**.*

2.1.1. Произведите поиск всех стихотворений, названия которых содержит только кириллицу.

2.1.2. Найдите все файлы с расширением **.jpeg**.

2.1.3. Найдите все файлы, которые были изменены за последние 20 минут.

2.1.4. Найдите все файлы, объемом больше 500 Кб.

2.2. Задание 2

*Примечание: используйте утилиту **sed**.*

2.2.1. Удалите все строки, начинающиеся с любой буквы из ваших инициалов. Если буквы повторяются, то добавьте еще любую другую букву.

2.2.2. Замените все символы после любой буквы из ваших инициалов на восклицательный знак.

2.2.3. Замените название у любого стихотворения на формат вида «Название, текущее время и ваши инициалы».

2.2.4. Выделите название стихотворение и автора в рамку из символов ‘-’ и ‘|’.

2.2.5. Замените все слова, начинающиеся на первую букву вашего имени на слово MIET.

2.2.6. Удалите все строки, начинающиеся с первой буквы вашей фамилии.

2.2.7. Переведите отечественные стихотворения на язык Тофслы и Вифслы (*) – после каждого слова добавьте окончание «сла».

2.2.8. Удалите все символы в каждой строке, после символа, совпадающего с первой буквой вашего имени. Если такой буквы в строке нет, то строка остается без изменения.

2.3.1. Задание 3

Примечание: комбинируйте различные утилиты.

2.3.1. Вычислите у скольких стихотворений вместо названия на первой строке в файле стоят “***”.

2.3.2. Выведите напротив каждого файла сообщение о том, содержит ли он восклицательный знак.

2.3.3. Рассчитайте, сколько раз в тексте стихотворений встречается предлог «на».

2.3.4. Вычислите самое часто встречающееся слово в монологе Гамлета.

Контрольные вопросы

1. Приведите пример задач, в которых использование утилит командной строки будет более рационально, чем использование графического интерфейса

2. Какие стандартные потоки вам известны? Покажите, как произвести вывод информации в один файл, а стандартный поток ошибок в другой? Как произвести вывод обоих потоков вывода в один файл?

3. Возможно ли одновременно вывести информацию на экран и в файл?

4. В чем отличие каналов ОС от потоков ввода/вывода?

5. В чём разница между find и grep.