

Основные возможности командного интерпретатора BASH

Типы команд и опции

```
command -a -bcd --word -o option1 --long-option=option2 argument1 argument2
```

- command
- -a
- -bcd
- --word
- -o option1
- --long-option=option2
- argument1

Командный интерпретатор GNU/Linux представляет собой специальную интерактивную программу. Эта программа обеспечивает для пользователей возможность запускать программы, управлять файлами в файловой системе, манипулировать процессами, которые исполняются в системе Linux, а также делать многое другое. Основой командного интерпретатора является командная строка. Командная строка — это интерактивная часть командного интерпретатора. Она позволяет вводить текстовые команды, после чего они интерпретируются и выполняются ядром.

Запускаемые команды могут быть внутренними (builtin), т.е. являться частью самого shell и внешними - отдельными исполняемыми файлами, появившимися в системе после установки соответствующих пакетов. Узнать, является ли команда внутренней или нет, можно с помощью команды "type" которая, в свою очередь также является внутренней командой. Сравните вывод команды:

```
[demo@localhost ~]$ type pwd pwd is a shell builtin
[demo@localhost ~]$ type hostname hostname is /bin/hostname [demo@localhost ~]$
type ls ls is aliased to 'ls --color=auto'
[demo@localhost ~]$ type quote
quote is a function quote ()
{ ...
}
```

В этом примере команда pwd является встроенной, команда hostname - внешней, а команда ls представляет собой псевдоним (т.е. на самом деле ссылается на некую другую команду, в данном случае на саму себя, но с параметром "--color"). Существует ещё четвёртая ситуация - команда может быть реализована в виде функции. О функциях речь пойдёт далее, пока можно просто отметить этот факт. Крайне полезным свойством командного интерпретатора является то, что он позволяет группировать команды, сохраняя их в файлах для последующего многократного выполнения. Такие файлы называются shell-сценарии.

В свою очередь, команды могут обладать параметрами, которые условно можно разделить на две категории. Это опции и аргументы. Опции меняют поведение самой программы. Например, команда "ls" запущенная без опций выводит список файлов в текущей директории, не показывая при этом скрытые файлы. Если же мы хотим видеть в выводе весь список файлов, включая скрытые, необходимо добавить к ней опцию "-a":

```
[demo@localhost ~]$ ls report.txt
[demo@localhost ~]$ ls -a
. .. .bash_logout .bash_profile .bashrc report.txt
```

В то время как аргументы указывают объект, к которому применяется команда, например:

```
[demo@localhost ~]$ ls report.txt
[demo@localhost ~]$ ls /home demo jsmith remote
```

И опций, и аргументов у команды может быть множество. Узнать, так ли это на самом деле и в какой очередности они должны следовать, можно с помощью команды man.

Раздел "SYNOPSIS" определяет порядок вызова команды. Квадратные скобки указывают на то, что данная часть команды является необязательной и может отсутствовать, многоточие говорит о том, что данная часть команды может повторяться многократно, т.е. опций и аргументов может быть сразу несколько. При этом, согласно стандарту, опции (если они не предполагают уточняющих элементов) могут вызываться в произвольном порядке:

```
[demo@localhost ~]$ ls -l -a
[demo@localhost ~]$ ls -a -l
[demo@localhost ~]$ ls -la
[demo@localhost ~]$ ls --all --long
```

Как видно из примера, опции реализованы в двух видах: длинном и коротком.

Короткий вариант быстрее набирать, в то время как длинный - проще запомнить (по крайней мере, человеку, знающему английский). Если опция вызывается в длинном варианте, перед ней ставится двойной дефис вместо одинарного. Это сделано для того, чтобы не было путаницы между опцией "help" и четырьмя опциями "h", "e", "l" и "p".

В один и тот же момент времени в системе могут параллельно сосуществовать сразу несколько командных интерпретаторов, причём их присутствие не мешает друг другу. Обычно список имеющихся в наличии интерпретаторов располагается в файле /etc/shells. При этом, один из интерпретаторов наделен особыми свойствами. Он называется «интерпретатор по умолчанию» и именно он запускается во всех случаях, когда пользователь успешно проходит аутентификацию в системе.

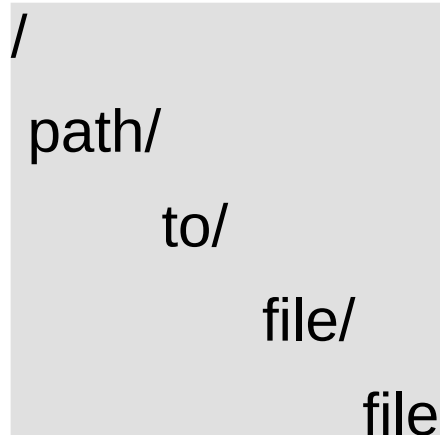
Узнать, какой интерпретатор используется в конкретно вашем случае, можно с помощью команды (её смысл мы разберём подробнее чуть ниже):

```
[demo@localhost ~]$ echo $SHELL
/bin/bash
```

Из вывода следует, что командный интерпретатор в нашем случае - это BASH. Название BASH - это аббревиатура от "Bourne-Again Shell", имя командного интерпретатора, ставшего оболочкой (в англоязычной литературе - shell) «de facto» во многих современных дистрибутивах Linux.

Навигация в дереве каталогов

- `cd /path`
- `cd to/file`
- `cd /path/to/file`
- `cat file`
- `cat /path/to/file/file`



```
graph TD; root[" / "] --> path[" path/ "]; path --> to[" to/ "]; to --> file1[" file/ "]; file1 --> file2[" file "];
```

The diagram illustrates a directory tree structure. It starts with a root directory represented by a slash (/). Below the root is a directory named 'path/'. Inside 'path/' is a directory named 'to/'. Inside 'to/' is a directory named 'file/'. Finally, inside 'file/' is a file named 'file'.

Ссылаясь на файл в какой-нибудь команде, можно обратиться к нему по абсолютному или относительному пути. Абсолютный путь - это полный путь к файлу, начиная с корня файловой системы. Он не зависит от нашего текущего местоположения в структуре директорий и ВСЕГДА начинается с «/», например:

```
[demo@localhost ~]$ cat /etc/sysconfig/network-scripts/ifcfg-lo
```

Если мы теперь перейдём в директорию `/etc/sysconfig`, то обратиться к тому же самому файлу мы сможем, используя относительный путь (т.е. путь, заданный относительно текущей директории):

```
[demo@localhost ~]$ cd /etc/sysconfig/
[demo@localhost sysconfig]$ cat network-scripts/ifcfg-lo
```

Две основные ошибки, которые допускают начинающие пользователи при указании пути к файлу:

1. Забывают ставить ведущий слеш при указании абсолютного пути. В этом случае shell будет считать, что путь относительный и попытаться найти директории и поддиректории в текущем каталоге
2. Наоборот, ставят слеш, подразумевая относительный путь. В этом случае shell считает, что путь абсолютный и пытается найти директории и поддиректории в корне.

Ссылаться по относительному пути можно, используя специальную конструкцию `".."`, означающую "уровень выше в иерархии директорий".

Конструкцию "точка-точка" можно применять в командах многократно, поднимаясь таким образом до самого корня.

Использование масок в именах

Маска	Описание	Пример
?	один любой произвольный символ	# ls -a /etc/m????s.*
*	любое произвольное количество символов (включая ноль символов)	# ls -a /etc/m????s.*
[abc]	любой символ, перечисленный в квадратных скобках	# ls /etc/[mnv]*.conf # ls /etc/[a-d]*.conf
[!abc]	любой символ, кроме перечисленных в скобках	# ls /etc/[!a-k]*.conf

При работе с файлами можно использовать маски. Маска - это способ сослаться на группу однотипных файлов: `ls -a /etc/m????s.*` - Отобразить все файлы из директории `/etc` удовлетворяющие следующим требованиям: файл должен начинаться с буквы `m`, затем идут четыре произвольных символа, буква `s`, точка и далее - произвольная строка.

Ну, и наконец, создать новую директорию можно с помощью команды `mkdir`:

```
[demo@localhost ~]$ mkdir /tmp/dir01
```

Каждый раз, когда речь идёт о работе с файлами или директориями, лежащими в домашнем каталоге пользователя, то вместо указания полного пути к домашнему каталогу (`/home/demo`) можно использовать символ "тильда" (`~`). Тильда всегда ссылается на домашний каталог пользователя. Например команды,

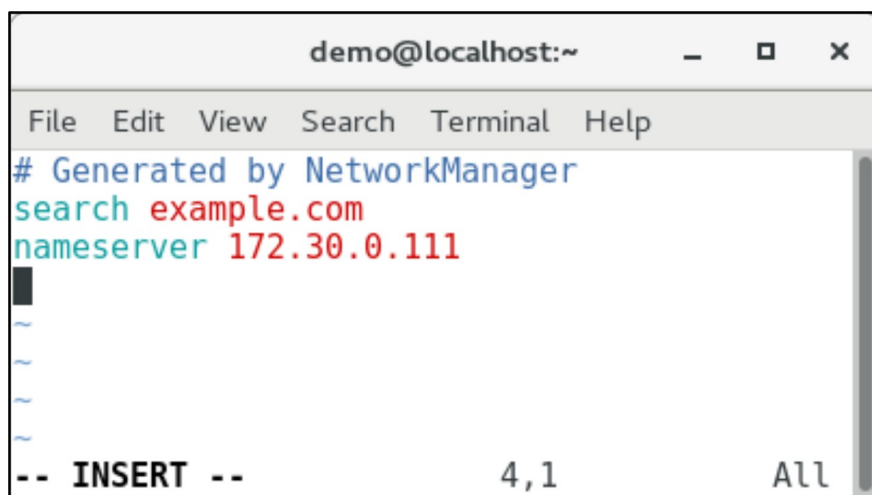
```
[demo@localhost ~]$ mkdir /home/demo/backup
```

и

```
[demo@localhost ~]$ mkdir ~/backup
```

делают одно и тоже, просто в первом случае домашний каталог указан явно, а во втором с помощью тильды. Тильда имеет два преимущества: во-первых, команды получаются короче, а во-вторых, если путь к домашнему каталогу пользователя измениться, то при использовании явного указания пути в сценарии придётся его редактировать, а при использовании тильды делать ничего не нужно: она автоматически будет ссылаться на новый каталог.

Редактор VIM



Редактор vim является улучшенной (VI iMproved) версией редактора vi - одного из старейших инструментов обработки текста, первая версия которого была создана в 1976 г.

Для чего необходимо уметь пользоваться редактором vim? Во-первых, это компактный редактор, не требующий каких-либо существенных ресурсов и способный работать в обстоятельствах, когда операционная система утратила возможности запускаться штатным образом (т.е. используется в аварийных ситуациях). Во-вторых, редактор vim неявным образом запускается некоторыми другими системными командами, требующими изменения настроек (скажем, при определении задач для системного планировщика). Кроме того, vim входит во все современные дистрибутивы, за исключением особо специфических.

В отличие от многих привычных редакторов, vim имеет модальный интерфейс. Это означает, что одни и те же клавиши в разных режимах работы выполняют разные действия. В редакторе vim есть несколько основных режимов: командный режим, режим редактирования, визуальный режим и режим исполнения (ex-mode).

Поначалу это вызывает некоторые затруднения в работе, но в дальнейшем вы обнаружите, что vim - мощный и быстрый редактор текстов.

По умолчанию, работа начинается в командном режиме. Это означает, что манипулировать текстом в привычном нам понимании (вставлять слова, строки, менять и удалять символы и т.д.) нельзя, т.к. vim ожидает ввода команд. Для переключения в режим редактирования нужно нажать кнопку «i» (Insert)

Теперь можно менять текст так, как мы привыкли делать это в традиционных редакторах. Режим обозначается надписью «- INSERT -» появляющейся внизу экрана.

Переключаться между вставкой и заменой в режиме редактирования можно кнопкой «Insert». Вернуться в режим команд можно нажатием кнопки «Esc».

Режим исполнения служит для манипуляций на файлом, как над объектом файловой системы (например, сохранение) а также для изменения параметров работы самого vim.

Команды VIM

Команда	Описание
:q	Выйти из редактора (можно использовать в случае, если текст не менялся)
:q!	Выйти из редактора с потерей сделанных изменений
:w	Сохранить файл на диск
:w имя_файла	Сохранить файл с указанным именем
:wq	Сохранить файл и выйти
:x	Сохранить файл и выйти (аналог wq)
:set number	Включить нумерование строк (удобно использовать при отладке сценариев, когда известен номер строки, вызывающей ошибку)
:set nonumber	Отключить нумерование строк. У редактора есть целый набор подобных переключателей, позволяющих влиять на его работу

В режиме команд можно также манипулировать содержимым файла, но его отличие от режима редактирования состоит в том, что операции осуществляются над отдельными компонентами текста (строкой, абзацем, предложением, словом, знаком и т.д.) например, команда 4dd означает удалить 4 строки, начиная с текущей.

Обратите внимание, что команды чувствительны к регистру и языку!

Команды VIM

Команда	Описание
h,j,k,l	Перемещение курсора влево, вниз, вверх и вправо
dd	Удалить строку
d\$	Удалить символы до конца строки
gg	Переместить курсор на первую строку файла
G	Переместить курсор на последнюю строку
yy	Скопировать строку в буфер
p	Вставить строки из буфера ниже курсора

P	Вставить строки из буфера выше курсора
u	«Undo» отмена последней команды
.	Повтор последней команды
/	Поиск по файлу вперёд
?	Поиск по файлу назад
n	Перемещение по найденному вперёд
N	Перемещение по найденному назад

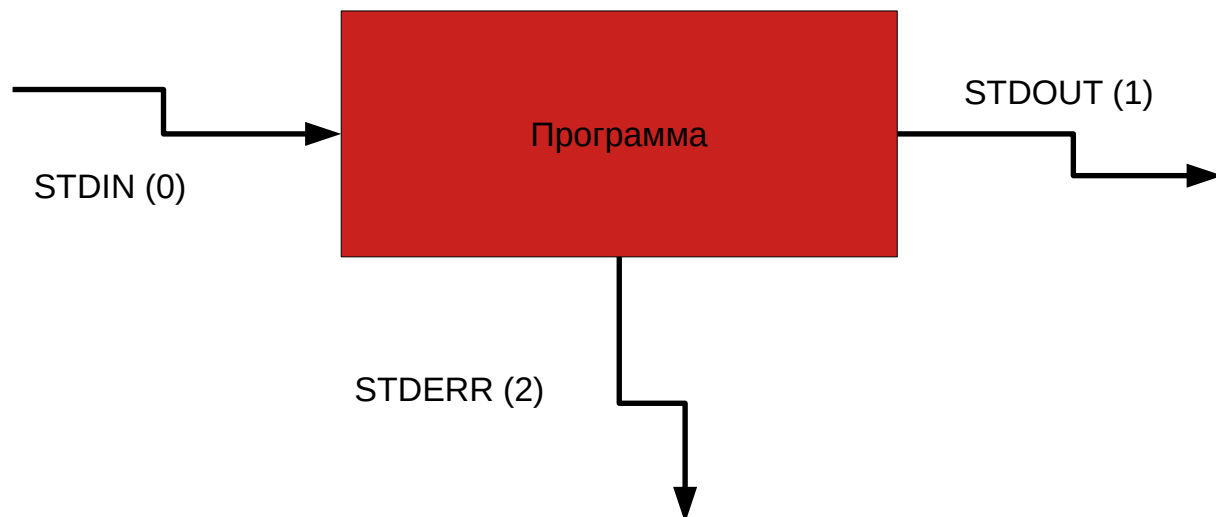
Вдобавок к описанным командам можно упомянуть возможность поиска с одновременной заменой. Синтаксис команды следующий:

`:[начальная_строка,конечная_строка]s/исходный_текст/конечный_текст/[g][i][c]`

Если замена требуется по всему файлу, то в качестве диапазона используется символ %.

- g - Заменять во всей строке (по-умолчанию, меняется только первое вхождение)
- i - Отключить чувствительность к регистру
- c - Запрашивать подтверждение перед заменой

Использование потоков ввода-вывода



Для взаимодействия данными работающие процессы используют механизм, называемый стандартными потоками ввода и вывода. Когда работающая программа выводит данные, мы считаем, что эти данные она выводит на экран. На самом деле программы выводят эти данные в специальный поток вывода, который по-умолчанию связан с экраном, поэтому и создается впечатление, что программа отображает результат на экране. Давайте рассмотрим, какие существуют потоки, как они работают и чем могут помочь администратору системы?

В Linux/UNIX системах разработчиками реализовано три стандартных потока: стандартный ввод (stdin), стандартный вывод (stdout) и стандартный вывод ошибок (stderr). У каждого потока есть свой номер (дескриптор), посредством которого на поток можно ссылаться:

Полезное свойство потоков заключается в том, что их можно перенаправлять.

Перенаправление стандартного ввода осуществляется с помощью символа «<», перенаправление стандартного вывода осуществляется с помощью символа «>» и, наконец, перенаправление стандартного вывода ошибок осуществляется с помощью символов «2>». Рассмотрим пример: выполним команду `ls -la /tmp`. В результате мы увидим содержимое директории /tmp. Теперь введём команду `ls -la /tmp > ls.txt`

Интерпретатор исполнил команду, но на экране не появилось ничего. Зато в текущей директории появился файл с именем `ls.txt`, содержащий результат работы команды `ls -la`. Это произошло потому, что мы перенаправили стандартный вывод в файл.

Давайте теперь выполним команду `date > ls.txt` и посмотрим на содержимое файла `ls.txt`. Как видно, оно теперь соответствует результату работы команды `date`. Т.е. при перенаправлении с помощью «>» если файл отсутствует, то он создается, а если файл уже существует, то происходит затирание его содержимого. Если требуется, чтобы в результате перенаправления имеющееся содержимое файла не было уничтожено, то для перенаправления вывода используется конструкция «>>»¹. В этом случае результат перенаправления будет добавлен в конец файла.

Теперь давайте рассмотрим вот такой пример: отобразим содержимое двух директорий, одна из которых существует, а другая - нет, после чего сделаем то же самое, но уже с перенаправлением вывода:

```
[demo@localhost ~]$ ls /wrongdir /home ls: cannot access /wrongdir: No such file or
directory /home:
demo elvis jsmith lfs student testuser
[demo@localhost ~]$
[demo@localhost ~]$ ls /wrongdir /home > /tmp/ls.txt ls: cannot access /wrongdir: No
such file or directory
```

Обратите внимание, что при выполнении второй команды результат работы команды `ls` для директории `/home` «исчез» (на самом деле он был перенаправлен в файл), а вот уведомление об ошибке - нет. Это произошло потому, что мы перенаправили только стандартный вывод. А стандартный вывод ошибок (куда команда `ls` отправляет уведомление об ошибке) так и остался связан с терминалом. Перенаправить оба потока можно следующим образом: `ls -la /wrongdir /home > ls.txt 2> ls.err` (в два разных файла) или `ls -la /wrongdir /home &>ls.all`

Перенаправление вывода нередко используют в shell-сценариях, когда желательно подавить вывод на экран, чтобы не смущать пользователя посторонними сообщениями. В этом случае, чтобы не создавать лишних файлов, перенаправляют вывод в специальное псевдоустройство, называемое `/dev/null`. Результат работы в этом случае перенаправляется в «никуда», при этом он нигде не сохраняется и теряется безвозвратно. Например:

```
cat /etc/passwd > /dev/null
```

Если требуется подавить вывод ошибок, то это можно сделать, перенаправив поток вывода ошибок в `/dev/null`:

```
cat /etc/s*.conf 2> /dev/null
```

(для того, чтобы убедиться в том, что при вызове были ошибки, попробуйте выполнить приведенную выше команду без перенаправления).

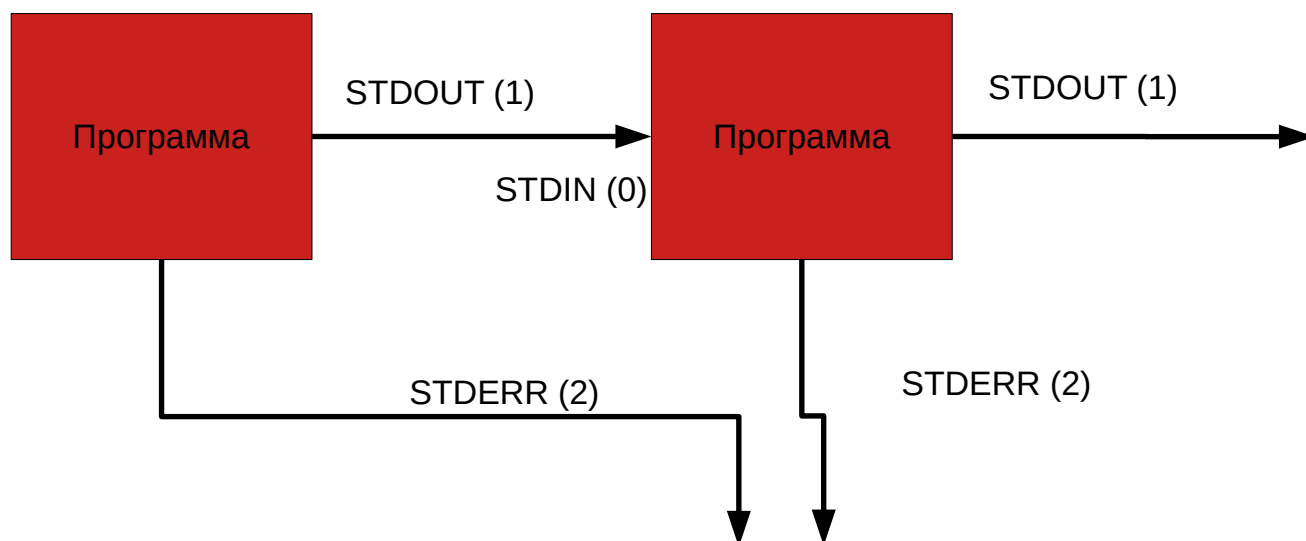
Перенаправление ввода осуществляется с помощью символа «<». Например:

```
mail -s "report" root < /tmp/ls.txt
```

Или пример перенаправления стандартного ввода и стандартного вывода в одной команде:

```
sort < /etc/passwd > /tmp/passwd.sorted
```

Конвейер



Стандартный вывод можно перенаправлять не только в файл, но и на ввод других программ. Этот механизм называется конвейеризация и осуществляется он с помощью символа «|», называемого в англоязычной литературе «pipe» (труба).

К примеру, нам необходимо подсчитать, сколько в системе установлено rpm-пакетов.

Для этого нам нужно получить их список и сосчитать, сколько в этом списке строк. Получить список установленных rpm-пакетов можно с помощью команды `rpm -qa`. Теперь остаётся только подсчитать количество строк в этом списке.

Сделать это можно с помощью команды `wc -l`. Если в качестве аргумента команде `wc` задан файл, она ведёт подсчет строк в этом файле (например: `wc -l /etc/passwd`). Если же имя файла не указано, то команда `wc` считает количество строк, поступивших из стандартного ввода, что нам и требуется:

```
rpm -qa | wc -l
```

Мы соединили вывод команды `rpm` с вводом команды `wc`. С помощью конвейера можно соединить не только две, но и три, четыре и более команд. Предположим, нам требуется получить список локальных пользователей системы, отсортированный по алфавиту и набранный заглавными буквами. Готовой утилиты, которая создает такой список, не существует. Однако, можно добиться результата, объединив в конвейер несколько простых утилит, каждая из которых решает собственную задачу. Например, в системе присутствует утилита `sort`, занимающаяся сортировкой строк, а также утилита `tr` (сокращение от `translate`), которая занимается преобразованием данных. Список пользователей системы можно получить из файла `/etc/passwd`:

```
cat /etc/passwd
```

Как мы видим, помимо имен пользователей в файле присутствует иная информация, от которой нам нужно избавиться для решения нашей задачи. Это можно сделать с помощью утилиты `cut`, позволяющей делить строки на составные части. Утилите `cut` нужно указать, какой символ нужно использовать в качестве разделителя и какую часть строки требуется оставить:

```
cat /etc/passwd | cut -d: -f1
```

Добавим сортировку:

```
cat /etc/passwd | cut -d: -f1 | sort
```

И преобразуем регистр:

```
cat /etc/passwd | cut -d: -f1 | sort | tr [a-z] [A-Z]
```

Таким образом, сочетая различные узкоспециализированные утилиты, можно получать удобные системные инструменты, которые не были изначально предусмотрены разработчиками

Если вы обратили внимание, то применение перенаправления вывода приводит к тому, что мы перестаём видеть результат работы программы. Нужно дожидаться ее завершения и просматривать содержимое файла, в который вывод был перенаправлен. Но что делать, если нужно одновременно писать данные в файл и наблюдать их на экране? Для этих целей используется специальная программа - «разветвитель» под названием `tee`:

```
ls -la /tmp | tee result.txt
```

Задание 1

- 1.Выполните команду `find /etc -name "*.conf" > /tmp/find.txt` Отобразились ли на экране какие-нибудь ошибки?
- 2.Теперь выполните команду так, чтобы результат её работы был перенаправлен в файл `/tmp/find.txt`, а ошибки - в файл `/tmp/find.err`
- 3.Повторите предыдущую команду так, чтобы и результат её работы, и ошибки были перенаправлены в файл `/tmp/find.all`
- 4.Выполните команду `date` так, чтобы результат её работы отобразился на экране и одновременно был записан в файл `/tmp/result`
- 5.Добавьте в файл `/tmp/result` результат работы команды `hostnamectl` так, чтобы все предыдущие данные, хранящиеся в этом файле, не пострадали. Отобразите содержимое файла на экране
- 6.С помощью команды `wc` подсчитайте количество всех процессов, запущенных в данный момент в системе. (Получить список процессов можно с помощью команды `"ps -e"`)

Ответы к заданию 1

Выполните команду `find /etc -name "*.conf" > /tmp/find.txt` Отобразились ли на экране какие-нибудь ошибки?

Теперь выполните команду так, чтобы результат её работы был перенаправлен в файл `/tmp/find.txt`, а ошибки - в файл `/tmp/find.err`

```
find /etc -name "*.conf" > /tmp/find.txt 2> /tmp/find.err
```

Повторите предыдущую команду так, чтобы и результат её работы, и ошибки были перенаправлены в файл `/tmp/find.all`

```
find /etc -name "*.conf" &> /tmp/find.all
```

Выполните команду `date` так, чтобы результат её работы отобразился на экране и одновременно был записан в файл `/tmp/result`

```
date | tee /tmp/result
```

Добавьте в файл `/tmp/result` результат работы команды `hostnamectl` так, чтобы все предыдущие данные, хранящиеся в этом файле, не пострадали. Отобразите содержимое файла на экране

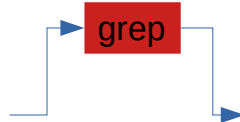
```
hostnamectl >> /tmp/result
```

С помощью команды `wc` подсчитайте количество всех процессов, запущенных в данный момент в системе. (Получить список процессов можно с помощью команды `"ps -e"`)

```
ps -e --no-headers | wc -l
```

grep

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do et dolore magna aliqua. Ut enim ad exercitation ullamco laboris nisi ut Duis aute irure dolor in cillum dolore eu fugiat nulla pariatur. non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

При обслуживании системы довольно часто возникает необходимость выбрать из потока каких-либо данных только те, которые соответствуют определённому шаблону. Допустим, нам нужно получить список установленных пакетов, в названии которых присутствует слово `system`. Сформировать список пакетов несложно, но это несколько сотен строк и выбирать среди них нужные - задача довольно непростая. Автоматизировать процесс позволяет команда `grep`:

```
[demo@localhost ~]$ grep 'root' /etc/passwd root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

Команда `grep` отбирает из файла `/etc/passwd` только те строчки, которые содержат слово `root`.

Опции grep

Опция	Описание
-n	Выводить номера строк, в которых присутствует искомое значение
-i	Отключить чувствительность к регистру символов
-A X	Выводить не только строку с найденным значением но и X строк, расположенных после неё (After)
-B X	Выводить не только строку с найденным значением но и X строк, расположенных до неё (Before)
-v	Инвертировать результат
-c	Вместо обнаруженных строк вывести их количество
-m X	Прекратить поиск после X совпадений

Регулярные выражения

Метасимвол	Значение	Пример
^	Начало строки	grep '^root' /etc/passwd
\$	Конец строки	grep 'bash\$' /etc/passwd
.	Любой одиночный символ, за исключением перевода строки	grep '^r...' /etc/passwd
*	Любое количество повторов предыдущего символа, включая ноль .* соответствует комбинации любых символов	grep 'hel*o' /usr/share/dict/words
[abc-f]	Подмножество символов. Означает, что символ может принимать одно из указанных в скобках значений: a,b,c,d,e,f	grep d[io]g /usr/share/dict/words
[^xyz]	Инвертированное подмножество. Означает, что символ может принимать	grep ^r[^o].* /etc/passwd
	любое значение, кроме перечисленного в скобках	

Обратите внимание, что `grep` выводит строки, в которых искомое значение (слово «root») встречается в любом месте. Но что делать, если требуется найти только те строки, в которых слово "root" находится в самом начале? В этом помогут регулярные выражения. Например:

```
[demo@localhost ~]$ grep '^root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Регулярные выражения (англ. *regular expressions*) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (англ. *wildcard characters*). По сути это строкаобразец (англ. *pattern*, по-русски её часто называют «шаблоном» или «маской»), состоящая из символов и метасимволов¹ и задающая правило поиска.

Примечание. Несмотря на то, что механизм регулярных выражений является универсальным и используется не только утилитой `grep`, но и в таких инструментах, как `awk`, `man`, `vim`, а также во многих языках программирования (PHP, Perl и др), его синтаксис не идентичен и может отличаться в разных реализациях.

Задание 2

- 1) Напишите регулярное выражение, с помощью которого можно получить строки, начинающиеся со слова «Start» либо «start»
- 2) Напишите регулярное выражение, с помощью которого можно получить строки, содержащие data4, data5, data6, data7, data8
- 3) Напишите регулярное выражение, с помощью которого можно получить строки, содержащие data2, data4, data6, data8
- 4) Напишите регулярное выражение, с помощью которого можно получить строки, заканчивающиеся на слово «finish.» (с точкой)
- 5) Напишите регулярное выражение, с помощью которого можно получить строку целиком: «From start to finish.»
- 6) Какие из перечисленных слов соответствуют регулярному выражению **back[un]p**
 - а) backup
 - б) backupp
 - в) backnp
 - г) backp
 - д)back
- 7) Используя регулярные выражения и команду grep, отобразите только тех пользователей из файла /etc/passwd, имена которых начинаются с буквы p
- 8) Используя регулярные выражения и команду grep, отобразите только тех пользователей из файла /etc/passwd, у которых shell по умолчанию - bash (т.е. последнее поле: /bin/bash)
- 9) Используя регулярные выражения и команду grep, отобразите только тех пользователей из файла /etc/passwd, у которых идентификатор пользователя или идентификатор группы (т.е. 3-е или 4-е поле) лежит в диапазоне от 10 до 19.
- 10) Отобразите содержимое файла /etc/selinux/config. Теперь, используя регулярные выражения и команду grep, отобразите все строки файла /etc/selinux/config не являющиеся комментариями.

Ответы к заданию 2

- Напишите регулярное выражение, с помощью которого можно получить строки, начинающиеся со слова «Start» либо «start»

`^[Ss]tart`

- Напишите регулярное выражение, с помощью которого можно получить строки data4 data5 data6 data7 data8

`data[4-8]`

- Напишите регулярное выражение, с помощью которого можно получить строки data2 data4 data6 data8

`data[2468]`

- Напишите регулярное выражение, с помощью которого можно получить строки, заканчивающиеся на слово «finish.»

`finish\.$`

- Напишите регулярное выражение, с помощью которого можно получить строку целиком: «From start to finish.»

`^From start to finish\.$`

- Какие из перечисленных слов соответствуют регулярному выражению **back[un]p**

а) backup

в) backnp

Используя регулярные выражения и команду grep, отобразите только тех пользователей из файла /etc/passwd, имена которых начинаются с буквы p

`$ grep ^p /etc/passwd`

Используя регулярные выражения и команду grep, отобразите только тех пользователей из файла /etc/passwd, у которых shell по умолчанию - bash (т.е. последнее поле: /bin/bash)

`$ grep bash$ /etc/passwd`

Используя регулярные выражения и команду `grep`, отобразите только тех пользователей из файла `/etc/passwd`, у которых идентификатор пользователя или идентификатор группы (т.е. 3-е или 4-е поле) лежит в диапазоне от 10 до 19.

```
$ grep ':1[0-9]:' /etc/passwd
```

Отобразите содержимое файла `/etc/selinux/config`. Теперь, используя регулярные выражения и команду `grep`, отобразите все строки файла

`/etc/selinux/config` не являющиеся комментариями `# cat /etc/selinux/config`

```
# grep -v '^#' /etc/selinux/config
```

второй вариант решения

```
# grep '^[^#]' /etc/selinux/config
```

Поиск файлов

`find [опции] путь [критерии_поиска] [действие]`

<code>find / -name fstab</code>	Найти во всей файловой системе файлы с именем <code>fstab</code>
<code>find /usr -iname "y*.conf"</code>	Найти в директории <code>/usr</code> и всех лежащих ниже поддиректориях файлы, с именем, начинающимся с <code>y</code> и расширением <code>.conf</code> без учёта регистра.

Довольно часто перед администратором системы возникает задача найти файлы, соответствующие каким-либо критериям. Например, найти файлы, размер которых больше 10 Гб или найти конфигурационные файлы, изменения в которые вносились в течении последних 24х часов (такое может потребоваться, если поведение системы изменилось само собой и администратор хочет убедиться, что никто ничего не менял в настройках), или найти все файлы, принадлежащие определённому пользователю и т.д. Все эти (и многие другие) действия можно осуществить с помощью команды `find`.

Параметры find

Критерий	Описание	Пример
-type	Тип искомого f - файл, d - директория, l - ссылка, s - сокет, p - канал, b - файл блочного устройства, c - файл символьного устройства	find / -type b
-user	Владелец файла, заданный именем	find / -user daemon
-group	Группа, связанная с файлом, заданная именем	find /var -group mail
-o	“или” (or) Позволяет задать несколько критериев	find / -name "*.bak" -o -name "*.tmp"
-not	“не” (not)	find /var -mmin -60 -not name messages
-uid	Владелец файла, заданный идентификатором	find / -uid 1000 -not -path "/home/*"
-gid	Группа, связанная с файлом, заданная идентификатором	find /home -gid 1000
-nouser	Файлы, имеющие владельцем неопределенного пользователя (возможно он был удалён)	find / -nouser

Параметры find

Критерий	Описание	Пример
-size	Файлы, имеющие определенный размер. Размер можно указывать в байтах (с), килобайтах (k), мегабайтах (M), гигабайтах (G) и 512байтных блоках (b)	find . -size +10M find ~ -size -100b find / -size 512c
-maxdepth	Указать глубину поиска. -maxdepth 1 означает «искать только в указанной директории»	сравните find /etc -maxdepth 1 -size -8c -type f и find /etc -maxdepth 2 -size -8c -type f
-mmin n	Файлы, изменённые n минут назад. - n означает найти файлы, изменённые в течении n минут.	find /etc -type f -mmin -30
-mtime n	Файлы, изменение которых последний раз было 24*n часов назад	find /var -mtime -1
-empty	Искать пустые файлы (в случае директории подразумевается, что в ней ничего нет)	find / -type d -empty

Еще несколько примеров использования find:

```
find /var/ -nouser -delete
```

Найти и удалить все файлы в директории /var и нижележащих, имеющие неопределённого владельца

```
find /home -type f -ls find /etc -type f -mtime -7 ! -mtime -3
```

Найти все файлы в директории /etc, которые были модифицированы за последние 7 суток, но не позднее, чем трое суток назад.

```
find /home -type f -exec cp {} /data \;
```

Найти все файлы в директории /home и скопировать их в директорию /data

Задание 3

- 1) Используя команду `find` определите, есть ли в вашей системе файлы, владельцем которых является пользователь `root`, а группа у которых - `mail`
- 2) Получите список всех файлов, лежащих в директории `/etc` и всех нижележащих директориях, размер которых более 100 кб.
- 3) С помощью какой команды можно осуществить автоматическое удаление всех пустых файлов `*.tmp` лежащих в директории `/tmp` и всех вложенных поддиректориях?

Используя команду `find` определите, есть ли в вашей системе файлы, владельцем которых является пользователь `root`, а группа у которых - `mail`

```
find / -user root -group mail -type f
```

Получите список всех файлов, лежащих в директории `/etc` и всех нижележащих директориях, размер которых более 100 кб.

```
find /etc -type f -size +100k
```

С помощью какой команды можно осуществить автоматическое удаление всех пустых файлов `*.tmp` лежащих в директории `/tmp` и всех вложенных поддиректориях?

```
find /tmp -name "*.tmp" -type f -empty -delete
```

sort и uniq

sort

сортирует строки

Может сортировать по определённому полю

uniq

Удаляет

повторяющиеся строки в отсортированном тексте

Аналог sort -u

Иногда возникает необходимость отсортировать какие-то данные по определённому алгоритму или исключить повторы в строках. Для этого в системе предусмотрены специальные инструменты. Это команды sort и uniq.

Команда sort осуществляет сортировку строк по разным алгоритмам. По умолчанию это сортировка в алфавитном порядке.

Если требуется отсортировать какие-то числовые данные, то у команды sort для этих целей предусмотрен специальный параметр "-n" (numeric). Например, получить список запущенных в системе процессов, отсортировав их по номеру идентификатора:

```
ps -e -o pid,comm | sort -n
```

Другая ситуация - исключить из данных, представленных в виде списка, повторяющиеся значения. Для этого в систему была добавлена команда uniq. Предположим, нам нужно узнать, от лица каких пользователей работают запущенные в системе процессы. Получить список процессов можно с помощью команды ps. У неё есть специальный параметр, позволяющий увидеть владельца процесса. Выглядеть эта команда будет так: "ps -e -o user,pid,comm". В таком виде она будет показывать владельца, идентификатор и имя команды, породившей процесс. Эти данные позволяют уже решить поставленную задачу, но вручную проанализировать получившийся список сложно и неоправданно долго. Поэтому, поступим следующим образом:

```
[demo@localhost ~]$ ps --no-headers -e -o user | sort | uniq
avahi
chrony
colord
dbus
...
```

Обратите внимание на параметр ps "--no-headers" — он убирает вывод заголовка. Без этого параметра результат окажется неверным, т.к. в вывод попадёт строка заголовка

sed

sed 's/user/jsmith/' /etc/passwd

p	печать (print)
d	d - удаление (delete)
s	s - подстановка (subtitution)
y	y - преобразование (transformation)
i	i - вставка текста перед заданной строкой
a	a - вставка текста после заданной строки (after)
c	c - замена строки целиком (change)

Как известно, для изменений, вносимых в текстовые файлы обычно используются текстовые редакторы. Но что делать, если требуется сделать изменения автоматически, не привлекая к этому процессу человека? Например, обновить конфигурационный файл сразу на нескольких хостах, используя shell-сценарий. Для этих целей можно использовать редактор sed. Сам по себе sed (также как и awk) не имеет отношения к интерпретатору BASH и является самостоятельной программой, но умение его применять - важное условие для создания высокопроизводительных сценариев.

Sed (от английского Stream EDitor) - это программа, представляющая собой потоковый текстовый редактор. Его особенность состоит в том, что он не подразумевает интерактивного редактирования файла (т.е. с помощью него нельзя открыть файл и что-то в нем поправить). При вызове sed, ему передаются инструкции, что именно требуется сделать и имя файла, над которым будут осуществляться манипуляции. Выглядит это, например, так:

```
[demo@localhost ~]$ sed 's/user/jsmith/' /etc/passwd > /tmp/passwd.new
```

Здесь редактор sed берёт строки из файла /etc/passwd и выводит их, подставляя (substitute, отсюда и буква s) первое вхождение подстроки «user» на подстроку «jsmith», в файл /tmp/passwd.new (по-умолчанию sed выводит результат своей работы в стандартный вывод). Если необходимо, чтобы изменения осуществлялись внутри исходного файла, необходимо использовать параметр “-i” (in place):

```
[demo@localhost ~]$ sed -i 's/jsmith/user/' /tmp/passwd.new
```


Команды sed

Операция	Описание
8d	Удалить восьмую строку
/^\$/d	Удалить все пустые строки
1,/^\$/d	Удалить все строки до первой пустой строки включительно
/root/d	Удалить все строки, содержащие подстроку root
/^root/d	Удалить все строки, начинающиеся с root
/John/p	Вывести все строки, содержащие подстроку John (нужна опция p)
s/^ */	Удалить пробелы, стоящие в начале строки
4i\test	Вставить слово "test" перед четвертой строкой

Давайте рассмотрим возможности sed. Перед каждой из команд может быть указан диапазон строк, к которому эта команда применяется, например:

```
[demo@localhost ~]$ seq 10 | sed '3,7d'
[demo@localhost ~]$ seq 10 | sed -n '3,7p'    # можно написать sed -n '3,+4p'
```

Параметр "-n" означает "тихий (silent)" вывод. Без него мы получим столбец цифр от 1 до 10, в котором цифры от 3 до 7 будут выведены дважды. С параметром -n будут выведены только цифры от 3 до 7. По-умолчанию sed выводит в стандартный вывод все передаваемые ему строки. Опция «-n» отключает этот вывод.

Если нужно обрабатывать файл с какой-то строки и до конца, то последняя строка задаётся символом "\$".

Результат работы команды sed можно инвертировать, добавив восклицательный знак, например:

```
[demo@localhost ~]$ seq 10 | sed -n '3,7!p'
```

Еще одна полезная опция: «-e». С её помощью можно задать несколько операций над потоком:

```
[demo@localhost ~]$ seq 20 | sed -e '3,5d' -e '10,12d'
```

Опция указывает, что следующая далее строка является инструкцией или набором инструкций. Вместо отдельной опции можно использовать точку с запятой:

```
[demo@localhost ~]$ seq 20 | sed '3,5d;10,12d'
```

Если инструкция содержит регулярное выражение, то указание диапазона строк требует заключения инструкции в фигурные скобки, например:

```
[demo@localhost ~]$ sed '2,5{/^$/d}' file01
```

Давайте ещё раз вернемся к операции замены. Для её применения предусмотрено несколько модификаторов, с помощью которых можно изменять поведение sed. Например, в самом первом примере редактор sed заменяет только самое первое вхождение подстроки «user», игнорируя последующие. Если требуется провести замену в строке целиком, то используется модификатор «g» (global). Сравните вывод следующих команд:

```
$ echo "Жил да был черный кот за углом. И кота ненавидел..." | sed 's/кот/котенок/'
Жил да был черный котенок за углом. И кота ненавидел..."
```

и

```
$ echo "Жил да был черный кот за углом. И кота ненавидел..." | sed
's/кот/котенок/g'
Жил да был черный котенок за углом. И котенка ненавидел..."
```

Вместо “g” может стоять целое число - номер заменяемой позиции.

Еще один модификатор - «i». С его помощью можно отключить чувствительность к регистру.

Инструкции, меняющие текст, sed может читать и из файла (для этого используется параметр “-f имя_файла”. Файл в этом случае просто содержит внутри команды, например:

```
s/jsmith/mtomson s/one/two
```

Вообще, возможности редактора sed гораздо шире. Важно понять принципы его работы и тогда не составит труда создавать сложные сценарии обработки текстовых данных.

awk

```
$ awk -F: '{print $1, "\t", $(NF-1)}' /etc/passwd
```

```
root      /root
```

```
bin       /bin
```

```
daemon    /sbin
```

```
...
```

AWK (а точнее, GNU AWK) – процессор обработки текстовых данных, функционал которого настолько велик, что может рассматриваться как самостоятельный язык программирования. В процессе работы с данными возможно:

- Объявлять переменные для хранения данных.
- Использовать арифметические и строковые операторы для работы с данными.
- Использовать структурные элементы и управляющие конструкции языка
- Создавать форматированные отчёты.

В простом случае awk можно запускать в виде отдельной команды, для сложных сценариев можно создавать файлы с набором команд, занимающихся манипуляциями с данными.

Awk воспринимает поступающие к нему данные в виде набора записей, а те, в свою очередь представляют собой наборы полей. Упрощенно, можно говорить о некоем обычном тексте, строки которого разделены символами перевода строки, запись — это строка. Поле — это слово в строке. По умолчанию, разделителем полей считается пробел, однако его можно переопределить с помощью опции «F». В общем случае, вызов awk выглядит так:

```
$ awk опции сценарий файл
```

Сценарий размещается внутри фигурных скобок, заключенных в кавычки (т. к. это текстовая строка). Например:

```
$ awk '{print "Welcome to awk"}'
```

Если запустить этот пример, то ничего не произойдет. Но это не значит, что сценарий не работает. Просто мы не указали ему никакого входного файла, и он ожидает данных из стандартного ввода stdin. Если сейчас ввести какую-нибудь строку текста и нажать Enter, мы увидим результат. При этом, что бы мы не ввели, результат будет одним и тем же. Для того, чтобы завершить работу сценария, нужно передать ему символ конца файла (End Of File, EOF). Сделать это можно, нажав комбинацию клавиш «Ctrl+d»

Полезным свойством `awk` является его способность делить текстовые строки на отдельные поля. Доступ к полям осуществляется через переменные `$1`, `$2`, `$3` итд. Переменная `$0` обозначает строку целиком, а переменная `$NF` — последнее поле в строке (удобно использовать, если количество полей переменное).

Пример:

```
$ awk -F: '{print $1, "\t", $(NF-1)}' /etc/passwd root      /root bin      /bin
daemon      /sbin
...
```

Скрипт отображает имя пользователя и его домашнюю директорию (предпоследнее поле в файле `passwd`)

Скрипт `awk` может состоять из нескольких строк. В этом случае они разделяются символом «;». Например:

```
$ echo "My name is $USER» | awk '{$4="root"; print $0}'
My name is root
```

Рис. 32 Использование разделителя в сценарии

Для многократного вызова `awk`-сценарии можно помещать в отдельные файлы.

Например, создадим файл `example01.awk` со следующим содержанием:

```
{
USER="user "
TEXT=" has a home directory at " print USER $1 TEXT $(NF-1)
}
```

Если мы пишем команды с новой строки, ставить точку с запятой не требуется.

Вызвать этот сценарий можно так:

```
awk -F: -f example01.awk /etc/passwd
```

Как и в большинстве языков программирования, в `awk` реализован условный оператор `"if"`. Предположим, нам нужно вывести имена пользователей, идентификаторы которых больше 999. Сделать это можно, например, так:

```
[ demo@localhost ~ ] $ awk -F: '{if ($3 > 999) print $1}' /etc/passwd nfsnobody user
```

Также `awk` умеет работать с тригонометрическими функциями и функциями для преобразования строк (например, сконвертировать все символы в верхний регистр), но одним из самых полезных свойств является возможность применения шаблонов к обрабатываемым строкам. Шаблоны представляют собой регулярные выражения, заключенные в символы `"/"`, например:

```
$ awk '/r..t/ {print $0}' /etc/passwd root:x:0:0:root:/root:/bin/bash operator:x:11:0:operator:/
root:/sbin/nologin ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

Задание 4

Изучите сценарий, находящийся в файле `/etc/cron.daily/mlocate` Что он делает? Используйте man-страницы неизвестных вам команд.

Примечание: Если файл `mlocate` отсутствует в вашей системе, выполните установку пакета: `"sudo yum install -y mlocate"`

Ответы на задание 4

Изучите сценарий, находящийся в файле `/etc/cron.daily/mlocate` Что он делает?

```
#!/bin/sh
nodevs=$(< /proc/filesystems awk '$1 == "nodev" && $2 != "rootfs" { print $2 }') renice
+19 -p $$ >/dev/null 2>&1 ionice -c2 -n7 -p $$ >/dev/null 2>&1
/usr/bin/updatedb -f "$nodevs"
```

Сценарий считывает файл `/proc/filesystems` содержащий список файловых систем, обслуживаемых в данный момент ядром. Из файла извлекается значение 2го поля для тех строк, где первое поле соответствует строке `nodev` и при этом второе - не `"rootfs"` . Считанные значения попадают в переменную `nodev` Далее сценарий понижает приоритет сам себе, а также понижает приоритет использования устройств ввода-вывода. Это сделано для того, чтобы его дальнейшая работа не сказывалась отрицательно на производительности системы. Далее вызывается команды `updatedb`, которая создает поисковую базу для команды `locate`. В качестве параметра ей передается значение переменной `nodev` - это список файловых систем, которые не нужно включать в поисковую базу.