Лабораторная работа №6 (DevOps №2). Права пользователей Linux, система управления конфигурациями Ansible, система управления версиями git

Оглавление

Подключение новых дисков к виртуальной машине

Выберите виртуальную машину для которой вы хотите добавить ещё один виртуальный диск и нажмите кнопку «Настройки» или сочетание клавиш **Ctrl+s**.

Перейдите на вкладку «**Носители**».

Кликните на «**Контроллер: SATA**» и затем на иконку «**Добавить жёсткий диск**».

Если вы перенесли виртуальный диск с другого компьютера и он отсутствует в этом списке, но уже существует, то нажмите на кнопку «Добавить» и укажите путь до него в файловой системе.

Если вы хотите создать новый диск, то нажмите кнопку «Создать».

Откроется знакомый по установке виртуальных машин мастер создания виртуальных дисков.

(!) Для выполнения задания с LVM понадобится добавить как минимум 1 новый неразмеченный виртуальный жесткий диск к виртуальной машине.

Менеджер логических томов (англ. logical volume manager) — подсистема операционных систем Linux и OS/2, позволяющая использовать разные области одного

жёсткого диска и/или области с разных жёстких дисков как один логический том. Реализована с помощью подсистемы device mapper.

LVM добавляет уровень абстракции между физическими/логическими дисками (привычными разделами, с которыми работает fdisk и аналогичные программы) и файловой системой. Это достигается путём разбивки изначальных разделов на блоки либо использования отдельных разделов или блочных устройств (physical volume (pv)) и объединения их в единый виртуальный том, точнее, группу томов (volume group (vg)), которая далее разбивается на логические тома (logical volume (lv)). Для файловой системы логический том представлен как обычное блочное устройство, хотя отдельные pv-тома могут находиться на разных физических устройствах (и даже сам pv может быть распределён подобно RAID).

Термины:

Физический том (англ. physical volume, pv) — устройство, представляющееся системе как один диск (жёсткий диск или его раздел, RAID-массив).

Группа томов (англ. volume group, vg) — несколько физических томов pv (группа, набор).

Логический том (англ. logical volume, lv) — логический раздел; аналог разделов hda1, sdb3 и др.; виртуальное блочное устройство.

Физический диапазон (англ. physical extent, pe) — область на физическом томе pv размером в несколько мегабайт. pv разбивается на области pe равного размера.

Логический диапазон (англ. logical extent, le) — область на логическом томе lv. lv разбивается на области le равного размера.

Основные команды для работы с LVM

Название команды	**Описание**	**Пример**
Pvs, vgs, lvs	Вывод информации об утилизации подключенных ФС	pvs
Pvcreate, pvremove	Подсчет размера файлов и директорий	pvcreate /dev/sda
vgcreate, vgremove	Изменение размера ФС ext2/ext3/ext4	vgcreate vg01 /dev/sda
lvcreate	Просмотр информации о разделах дисков	lvcreate -n lv01 -l 100%VG vg01
lvextend	Запуск проверки ФС на целостность	lvextend /dev/vg01/lv01 -L+1000M
pvmove	Создание ФС на диске	pvmove /dev/sda /dev/sdb

Более подробно работу с LVM можно увидеть на примере: https://www.dmosk.ru/instruktions.php?object=lvm

Настройка отчетов LVM

С помощью **pvs**, **lvs**, **vgs** можно создавать отчеты о состоянии объектов LVM. Каждая строка в отчете содержит информацию об одном объекте. Вывод можно отфильтровать по физическим томам, по группе томов, по логическим томам, сегментам физических или логических томов.

Изменение формата

Независимо от того, используете ли вы pvs, lvs или vgs, выбранная команда по умолчанию отображает стандартный набор полей, что можно переопределить с помощью различных опций.

Аргумент -o позволяет выбрать поля для вывода. Например, стандартный вывод команды pvs выглядит так:

pvs

```
PV         VG     Fmt  Attr PSize  PFree
/dev/sdb1  new_vg lvm2 a-   17.14G 17.14G
/dev/sdc1  new_vg lvm2 a-   17.14G 17.09G
```

Следующая команда покажет только имя и размер физических томов.

pvs -o pv_name,pv_size

```
PV         PSize
/dev/sdb1  17.14G
/dev/sdc1  17.14G
```

Дополнительное поле можно добавить с помощью знака "+" в комбинации с "-o".

Пример добавления идентификатора UUID в стандартный отчет:

pvs -o +pv_uuid

```
PV         VG     Fmt  Attr PSize  PFree  PV UUID
/dev/sdb1 new_vg lvm2 a- 17.14G 17.14G onFF2w-1fLC-ughJ-D9eB-M7iv-6XqA-dqGeXY
/dev/sdc1 new_vg lvm2 a- 17.14G 17.09G Joqlch-yWSj-kuEn-IdwM-01S9-X08M-mcpsVe
```

--noheadings спрячет строку заголовков, что используется при создании сценариев.

Следующий пример использует аргумент --noheadings в комбинации с pv_name для отображения списка всех физических томов.

pvs --noheadings -o pv_name

/dev/sdb1

/dev/sdc1

--separator разделитель позволяет отделить поля друг от друга.

В следующем примере поля вывода команды pvs разделены знаком равенства.

pvs --separator =

PV=VG=Fmt=Attr=PSize=PFree

/dev/sdb1=new_vg=lvm2=a-=17.14G=17.14G

/dev/sdc1=new_vg=lvm2=a-=17.14G=17.09G

Для выравнивания полей при использовании разделителя, можно дополнительно указать --aligned.

pvs --separator = --aligned

PV =VG =Fmt =Attr=PSize =PFree

/dev/sdb1 =new_vg=lvm2=a- =17.14G=17.14G

/dev/sdc1 =new_vg=lvm2=a- =17.14G=17.09G

Для добавления в отчет списка неисправных томов используется параметр -P команд lvs и vgs (см. Раздел 6.2, «Получение информации о неисправных устройствах»).

Справочные страницы pvs, vgs и lvs содержат полный перечень параметров.

Поля группы томов могут быть смешаны с полями физического или логического тома (или их сегментов), но поля физического тома не могут быть смешаны с полями логического тома. Например, следующая команда покажет по одному физическому тому в строке:

vgs -o +pv_name

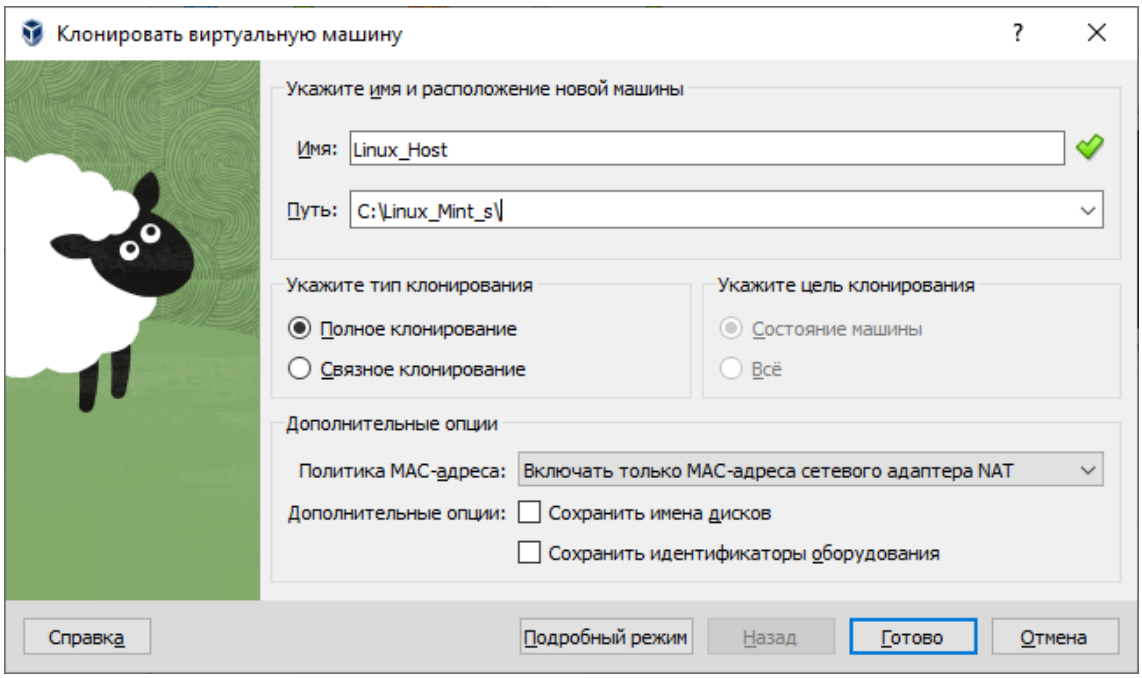
VG #PV #LV #SN Attr VSize VFree PV

new_vg 3 1 0 wz--n- 51.42G 51.37G /dev/sdc1

new_vg 3 1 0 wz--n- 51.42G 51.37G /dev/sdd1

Создание клона виртуальной машины

Чтобы открыть окно создания клона виртуальной машины, необходимо перейти в меню «Машина» - «Клонировать».



Настройка сети между виртуальными машинами в VirtualBox

Сеть NAT объединяет виртуальные машины в локальную сеть. Как и в случае с обычным NAT, у каждой есть доступ в интернет, но от доступа извне они изолированы.

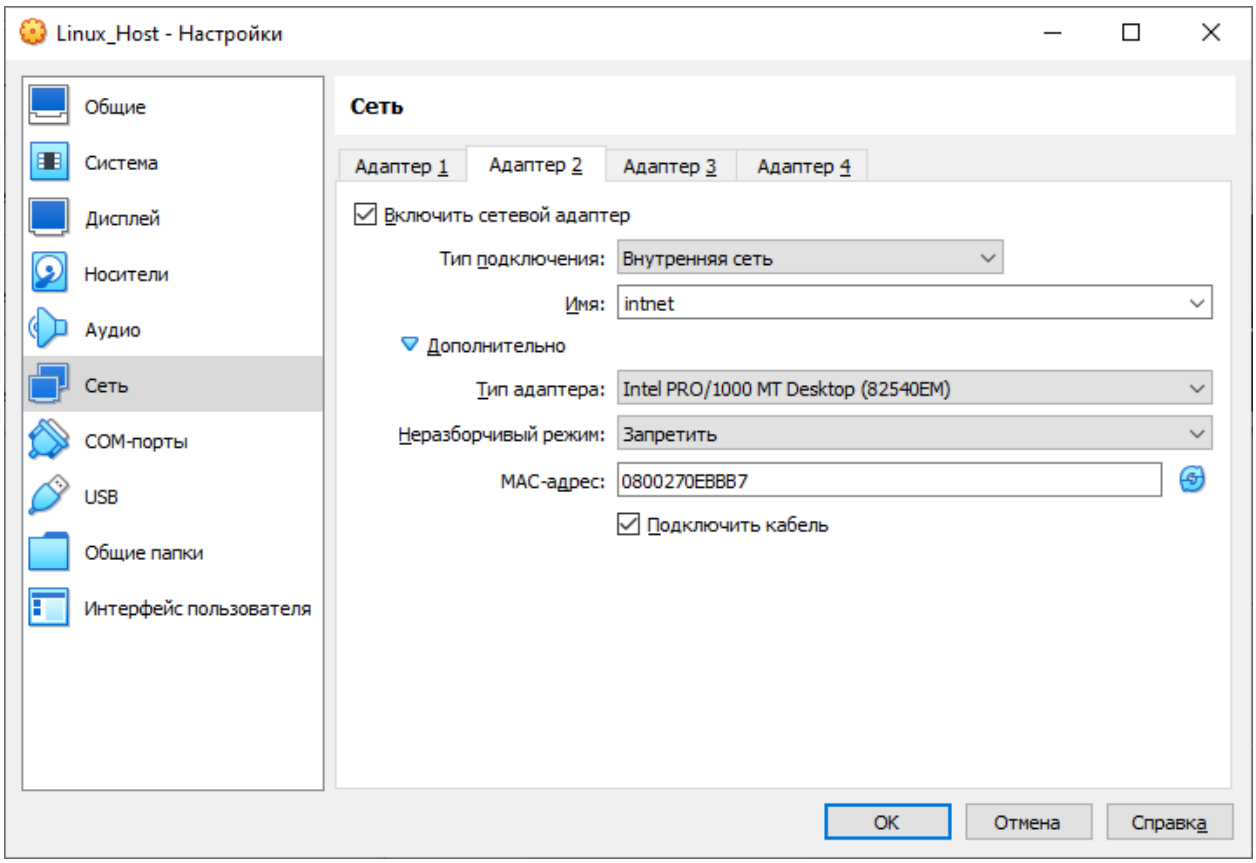
Чтобы создать сеть из виртуальных машин VirtualBox, есть 2 варианта: **внутренняя сеть** и **виртуальный адаптер хоста** (рекомендуемый вариант).

Режим подключения: внутренняя сеть

Этот тип соединения полезен, если мы хотим получить максимальную защиту от внешних вторжений для нашей виртуальной машины.

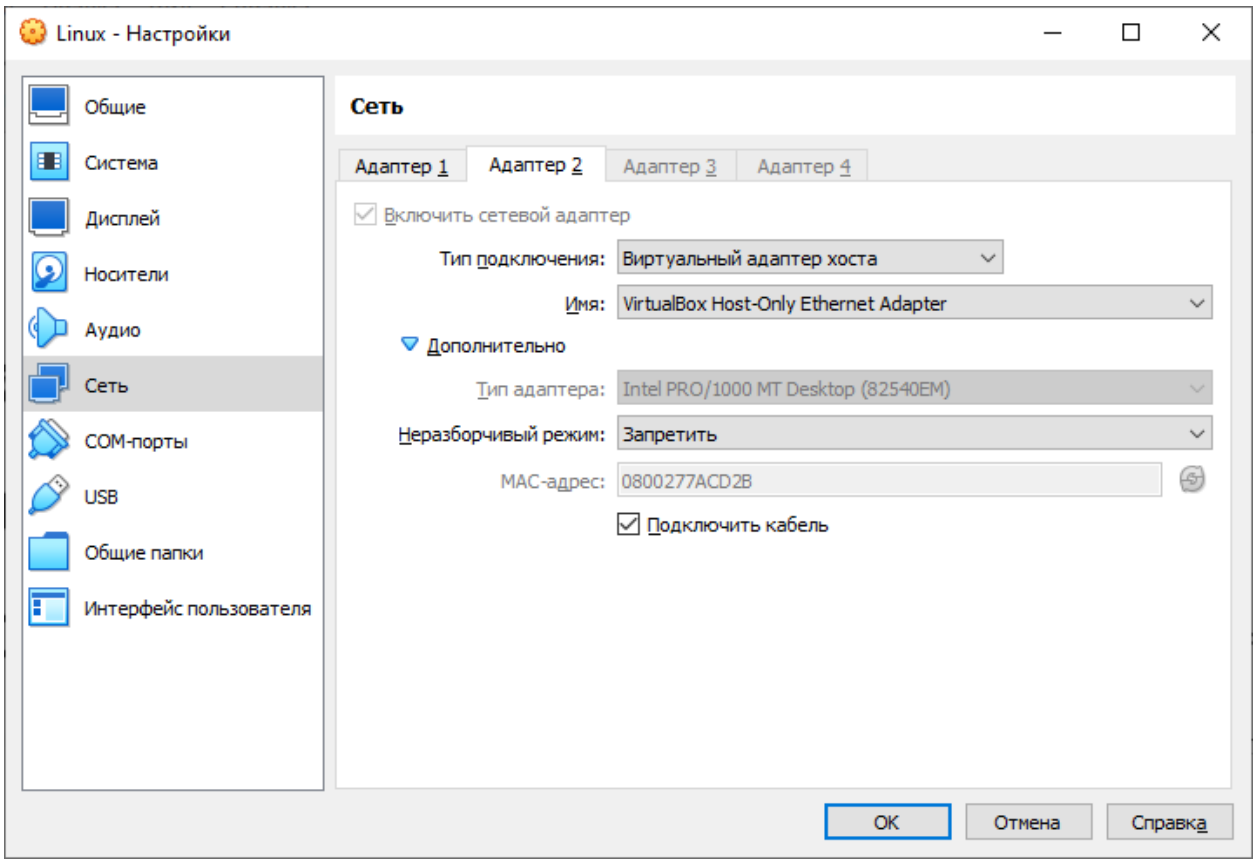
Благодаря этому режиму **можно обмениваться данными между виртуальными машинами**, как если бы это была сеть LAN, **но у нас НЕ будет доступа ни к Интернету** (внешней сети), ни к хост-компьютерам.

Если мы посмотрим на сетевое соединение операционной системы, мы увидим, что у нас нет шлюза, и у нас даже не будет IP-адреса, аналогичного адресу нашего хост-компьютера.



Виртуальный адаптер хоста (Host-only)

При подключении типа «Виртуальный адаптер хоста» гостевые ОС могут взаимодействовать между собой, а также с хостом. Но все это только внутри самой виртуальной машины VirtualBox. В этом режиме адаптер хоста использует свое собственное, специально для этого предназначенное устройство, которое называется vboxnet0. Также им создается под-сеть и назначаются IP-адреса сетевым картам гостевых операционных систем. Гостевые ОС не могут взаимодействовать с устройствами, находящимися во внешней сети, так как они не подключены к ней через физический интерфейс. Режим «Виртуальный адаптер хоста» предоставляет ограниченный набор служб, полезных для создания частных сетей под VirtualBox для ее гостевых ОС.



В отличие от других продуктов виртуализации, адаптер, работающий под протоколом NAT в VirtualBox, не может выступать в роли связующего моста между сетевым устройством по умолчанию на хостах. Поэтому невозможен прямой доступ извне к машинам, "спрятанным" за NAT - ни к программам, работающим на них; ни к данным, находящимся на самих хостах.

Как правило, хост имеет свой собственный сетевой адрес, который используется для выхода в Интернет. Обычно это 192.168.0.101. В режиме «Виртуальный адаптер хоста» машина-хост также выступает в роли роутера VirtualBox и обладает IP-адресом по умолчанию 192.168.56.1. Создается внутренняя локальная сеть, обслуживающая все гостевые операционные системы, настроенные для режима «Виртуальный адаптер хоста» и видимые для остальной части физической сети. Адаптер vboxnet0 использует адреса из диапазона, начинающегося с 192.168.56.101. Но при желании можно изменить адрес по умолчанию.

Подобно адаптеру в режиме «Сетевой мост», в режиме «Виртуальный адаптер хоста» используются разные диапазоны адресов. Можно легко настроить гостевые системы для получения IP-адресов, используя для этого встроенный DHCP-сервер виртуальной машины VirtualBox.

В дополнение нужно сказать, что в режиме «Виртуальный адаптер хоста» созданная им сеть не имеет внешнего шлюза для выхода в Интернет, как для хоста, так и для гостевых операционных систем. Он работает только как обычный сетевой коммутатор, соединяя между собой хост и гостевые системы. Поэтому адаптер в режиме «Виртуальный адаптер хоста» не предоставляет гостевым машинам выход в Интернет; vboxnet0 по умолчанию не имеет шлюза. Дополнительные возможности для этого

адаптера значительно упрощают настройку сети между хостом и гостевыми ОС, однако все же отсутствует внешний доступ или перенаправление портов.

(!) Для взаимодействия между машинами «Виртуальный адаптер хоста» должен быть настроен на всех виртуальных машинах.

Более подробно: http://rus-linux.net/MyLDP/vm/VirtualBox-networking.html

Перенаправление ввода/вывода в Linux

Стандартные потоки ввода и вывода в Linux являются одним из наиболее распространенных средств для обмена информацией процессов, а перенаправление >, >> и | является одной из самых популярных конструкций командного интерпретатора.

Стандартный ввод при работе пользователя в терминале передается через клавиатуру.

Стандартный вывод и стандартная ошибка отображаются на дисплее терминала пользователя в виде текста.

Ввод и вывод распределяется между тремя стандартными потоками:

- stdin — стандартный ввод (клавиатура),
- stdout — стандартный вывод (экран),
- stderr — стандартная ошибка (вывод ошибок на экран).

Потоки также пронумерованы:

- stdin — 0,
- stdout — 1,
- stderr — 2.

Из стандартного ввода команда может только считывать данные, а два других потока могут использоваться только для записи. Данные выводятся на экран и считываются с клавиатуры, так как стандартные потоки по умолчанию ассоциированы с терминалом пользователя. Потоки можно подключать к чему угодно: к файлам, программам и даже устройствам. В командном интерпретаторе bash такая операция называется перенаправлением:

< file — использовать файл как источник данных для стандартного потока ввода.

> file — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху.

2> file — направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху.

>>file — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.

2>>file — направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.

&>file или >&file — направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >file 2>&1.

Стандартный ввод

Стандартный входной поток обычно переносит данные от пользователя к программе. Программы, которые предполагают стандартный ввод, обычно получают входные данные от устройства типа клавиатура. Стандартный ввод прекращается по достижении **EOF** (конец файла), который указывает на то, что данных для чтения больше нет.

EOF вводится нажатием сочетания клавиш **Ctrl+D**.

Рассмотрим работу со стандартным выводом на примере команды **cat**.

cat обычно используется для объединения содержимого двух файлов.

cat отправляет полученные входные данные на дисплей терминала в качестве стандартного вывода и останавливается после того как получает **EOF**.

В открывшейся строке введите, например, **1** и нажмите клавишу **Enter**. На дисплей выводится **1**. Введите **a** и нажмите клавишу **Enter**. На дисплей выводится **a**.

```
cat
1
1
a
a
```

Для завершения ввода данных следует нажать сочетание клавиш **Ctrl + D**.

Стандартный вывод

Стандартный вывод записывает данные, сгенерированные программой. Когда стандартный выходной поток не перенаправляется в какой-либо файл, он выводит текст на дисплей терминала.

При использовании без каких-либо дополнительных опций, команда **echo** выводит на экран любой аргумент, который передается ему в командной строке:

```
echo Пример
Пример
```

При выполнении **echo** без каких-либо аргументов, возвращается пустая строка.

Команда объединяет три файла: file1, file2 и file3 в один файл bigfile:

```
cat file1 file1 file1 > bigfile
```

Команда **cat** по очереди выводит содержимое файлов, перечисленных в качестве параметров на стандартный поток вывода. Стандартный поток вывода перенаправлен в файл **bigfile**.

Стандартная ошибка

Стандартная ошибка записывает ошибки, возникающие в ходе исполнения программы. Как и в случае стандартного вывода, по умолчанию этот поток выводится на терминал дисплея.

Рассмотрим пример стандартной ошибки с помощью команды **ls**, которая выводит список содержимого каталогов.

При запуске без аргументов **ls** выводит содержимое в пределах текущего каталога.

Введем команду **ls** с каталогом **%** в качестве аргумента:

```
ls %
```

В результате должно выводиться содержимое соответствующей папки. Но так как каталога **%** не существует, на дисплей терминала будет выведен следующий текст стандартной ошибки:

```
ls: cannot access %: No such file or directory
```

Перенаправление потока

Linux включает в себя команды перенаправления для каждого потока.

Команды со знаками > или < означают перезапись существующего содержимого файла:

- > — стандартный вывод,
- < — стандартный ввод,
- 2> — стандартная ошибка.

Команды со знаками >> или << не перезаписывают существующее содержимое файла, а присоединяют данные к нему:

- >> — стандартный вывод,
- << — стандартный ввод,
- 2>> — стандартная ошибка.

В приведенном примере команда cat используется для записи в файл **file1**, который создается в результате цикла:

```
cat > file1
```

```
a
b
c
```

Для завершения цикла нажмите сочетание клавиш **Ctrl + D**.

Если файла **file1** не существует, то в текущем каталоге создается новый файл с таким именем.

Для просмотра содержимого файла **file1** введите команду:

```
cat file1
```

В результате на дисплей терминала должно быть выведено следующее:

```
a
b
c
```

Для добавления нового текста к уже существующему в файле с помощью двойных скобок >> выполните команду:

```
cat >> file1
a
b
c
```

Для завершения цикла нажмите сочетание клавиш **Ctrl + D**.

Каналы

Каналы используются для перенаправления потока из одной программы в другую. Стандартный вывод данных после выполнения одной команды перенаправляется в другую через канал. Данные первой программы, которые получает вторая программа, не будут отображаться. На дисплей терминала будут выведены только отфильтрованные данные, возвращаемые второй командой.

Пример:

```
ls | less
```

В результате каждый файл текущего каталога будет размещен на новой строке.

Перенаправлять данные с помощью каналов можно как из одной команды в другую, так и из одного файла к другому, а перенаправление с помощью > и >> возможно только для перенаправления данных в файлах.

Для сохранения имен файлов, содержащих строку «LOG», используется следующая команда:

```
dir /catalog | find "LOG" > loglist
```

Вывод команды **dir** отсылается в команду-фильтр **find**. Имена файлов, содержащие строку «**LOG**», хранятся в файле **loglist** в виде списка (например, **Config.log**, **Logdat.svd** и **Mylog.bat**).

При использовании нескольких фильтров в одной команде рекомендуется разделять их с помощью знака канала |.

Фильтры

Фильтры представляют собой стандартные команды Linux, которые могут быть использованы без каналов:

find — возвращает файлы с именами, которые соответствуют передаваемому аргументу.

grep — возвращает только строки, содержащие (или не содержащие) заданное регулярное выражение.

tee — перенаправляет стандартный ввод как стандартный вывод и один или несколько файлов.

tr — находит и заменяет одну строку другой.

wc — подсчитывает символы, линии и слова.

Как правило, все нижеприведенные команды работают как фильтры, если у них нет аргументов (опции могут быть).

cat — считывает данные со стандартного потока ввода и передает их на стандартный поток вывода. Без опций работает как простой повторитель. С опциями может фильтровать пустые строки, нумеровать строки и делать другую подобную работу.

head — показывает первые 10 строк (или другое заданное количество), считанных со стандартного потока ввода.

tail — показывает последние 10 строк (или другое заданное количество), считанные со стандартного потока ввода. Важный частный случай tail -f, который в режиме слежения показывает концовку файла. Это используется, в частности, для просмотра файлов журнальных сообщений.

cut — вырезает столбец (по символам или полям) из потока ввода и передает на поток вывода. В качестве разделителей полей могут использоваться любые символы.

sort — сортирует данные в соответствии с какими-либо критериями, например, арифметически по второму столбцу.

uniq — удаляет повторяющиеся строки. Или (с ключом -c) не просто удалить, а написать сколько таких строк было. Учитываются только подряд идущие одинаковые строки, поэтому часто данные сортируются перед тем как отправить их на вход программе.

bc — вычисляет каждую отдельную строку потока и записывает вместо нее результат вычисления.

hexdump — показывает шестнадцатеричное представление данных, поступающих на стандартный поток ввода.

strings — выделяет и показывает в стандартном потоке (или файле) то, что напоминает строки. Всё что не похоже на строковые последовательности, игнорируется. Команда полезна в сочетании с grep для поиска интересующих строковых последовательностей в бинарных файлах.

sed — обрабатывает текст в соответствии с заданным скриптом. Наиболее часто используется для замены текста в потоке.

awk — обрабатывает текст в соответствии с заданным скриптом. Как правило, используется для обработки текстовых таблиц, например, вывод ps aux и т.д.

sh -s — текст, который передается на стандартный поток ввода **sh -s**. может интерпретироваться как последовательность команд **shell**. На выход передается результат их исполнения.

ssh — средство удаленного доступа **ssh**, может работать как фильтр, который подхватывает данные, переданные ему на стандартный поток ввода, затем передает их на удаленный хост и подает на вход процессу программы, имя которой было передано ему в качестве аргумента. Результат выполнения программы (то есть то, что она выдала на стандартный поток вывода) передается со стандартного вывода **ssh**.

Если в качестве аргумента передается файл, команда-фильтр считывает данные из этого файла, а не со стандартного потока ввода (есть исключения, например, команда **tr**, обрабатывающая данные, поступающие исключительно через стандартный поток ввода).

Команда **tee**, как правило, используется для просмотра выводимого содержимого при одновременном сохранении его в файл.

```
wc ~/stream | tee file2
```

Допускается перенаправление нескольких потоков в один файл:

```
ls -z >> file3 2>&1
```

В результате сообщение о неверной опции «**z**» в команде **ls** будет записано в файл **t2**, поскольку **stderr** перенаправлен в файл.

Для просмотра содержимого файла **file3** введите команду **cat file3**.

В результате на дисплее терминала отобразиться следующее:

```
ls: invalid option -- 'z'

Try 'ls --help' for more information.
```

Коды завершения

В Linux и других Unix-подобных операционных системах программы во время завершения могут передавать значение родительскому процессу. Это значение называется кодом завершения или состоянием завершения. В POSIX по соглашению действует стандарт: программа передаёт 0 при успешном исполнении и 1 или большее число при неудачном исполнении.

Почему это важно? Если смотреть на коды завершения в контексте скриптов для командной строки, ответ очевиден. Любой полезный Bash-скрипт неизбежно будет использоваться в других скриптах или его обернут в однострочник Bash. Это особенно актуально при использовании инструментов автоматизации типа **SaltStack** или инструментов мониторинга типа **Nagios**. Эти программы исполняют скрипт и проверяют статус завершения, чтобы определить, было ли исполнение успешным.

Кроме того, даже если вы не определяете коды завершения, они всё равно есть в ваших скриптах. Но без корректного определения кодов выхода можно столкнуться с проблемами: ложными сообщениями об успешном исполнении, которые могут повлиять на работу скрипта.

Что происходит, когда коды завершения не определены

В Linux любой код, запущенный в командной строке, имеет код завершения. Если код завершения не определён, Bash-скрипты используют код выхода последней запущенной команды. Чтобы лучше понять суть, обратите внимание на пример.

```
#!/bin/bash

touch /root/test

echo created file
```

Этот скрипт запускает команды **touch** и **echo**. Если запустить этот скрипт без прав суперпользователя, команда **touch** не выполнится. В этот момент мы хотели бы получить информацию об ошибке с помощью соответствующего кода завершения. Чтобы проверить код выхода, достаточно ввести в командную строку специальную переменную **$?**. Она печатает код возврата последней запущенной команды.

```
$ ./tmp.sh

touch: cannot touch '/root/test': Permission denied

created file

$ echo $?

0
```

Как видно, после запуска команды **./tmp.sh** получаем код завершения **0**. Этот код говорит об успешном выполнении команды, хотя на самом деле команда не выполнилась. Скрипт из примера выше исполняет две команды: **touch** и **echo**. Поскольку код завершения не определён, получаем код выхода последней запущенной команды. Это команда **echo**, которая успешно выполнилась.

Если убрать из скрипта команду **echo**, можно получить код завершения команды touch.

Поскольку **touch** в данном случае — последняя запущенная команда, и она не выполнилась, получаем код возврата **1**.

Проверка кода завершения

Выше мы пользовались специальной переменной **$?**, чтобы получить код завершения скрипта. Также с помощью этой переменной можно проверить, выполнилась ли команда **touch** успешно.

```
#!/bin/bash

touch /root/test 2> /dev/null

if [ $? -eq 0 ]

then

        echo "Successfully created file"

else

        echo "Could not create file" >&2

fi
```

После рефакторинга скрипта получаем такое поведение:

- Если команда **touch** выполняется с кодом 0, скрипт с помощью **echo** сообщает об успешно созданном файле.
- Если команда **touch** выполняется с другим кодом, скрипт сообщает, что не смог создать файл.

Любой код завершения кроме 0 значит неудачную попытку создать файл. Скрипт с помощью **echo** отправляет сообщение о неудаче в **stderr**.

```
$ ./tmp.sh

Could not create file
```

Собственный код завершения

Наш скрипт уже сообщает об ошибке, если команда **touch** выполняется с ошибкой. Но в случае успешного выполнения команды мы всё также получаем код 0.

Поскольку скрипт завершился с ошибкой, было бы не очень хорошей идеей передавать код успешного завершения в другую программу, которая использует этот скрипт. Чтобы добавить собственный код завершения, можно воспользоваться командой **exit**.

```
#!/bin/bash
touch /root/test 2> /dev/null
if [ $? -eq 0 ]
then
        echo "Successfully created file"
        exit 0
else
        echo "Could not create file" >&2
        exit 1
fi
```

Теперь в случае успешного выполнения команды **touch** скрипт с помощью **echo** сообщает об успехе и завершается с кодом 0. В противном случае скрипт печатает сообщение об ошибке при попытке создать файл и завершается с кодом 1.

```
$ ./tmp.sh
Could not create file
$ echo $?
1
```

Пакетные менеджеры

Большинство систем управления пакетами строятся на наборах файлов пакетов. Файл пакета – это, как правило, архив, который содержит скомпилированные бинарные файлы, скрипты установки и другие ресурсы, составляющие программу. Также пакеты содержат ценные метаданные, в том числе их зависимости (список пакетов, необходимых для запуска программы).

CentOS, **Fedora** и другие системы **Red Hat** используют файлы **RPM**. В **CentOS** для взаимодействия с пакетами и репозиториями используется менеджер **yum**. В последних версиях **Fedora yum** был заменён модернизированным менеджером **dnf**. В системе **Debian** и основанных на ней системах (**Ubuntu**, **Linux Mint**, **Raspbian**) используется формат .**deb**. Пакетный менеджер **APT** (Advanced Packaging Tool) предоставляет команды, используемые для наиболее распространенных операций: поиска репозиториев, управления обновлениями, установки набора пакетов и их зависимостей. Команды **APT**

работают как фронтэнд утилиты нижнего уровня **dpkg**, которая обрабатывает установку индивидуальных пакетов .**deb** на локальную систему; при необходимости эту утилиту можно вызывать явно.

Системой бинарных файлов **FreeBSD** управляет команда **pkg**. Кроме того, **FreeBSD** предоставляет коллекцию портов, локальную структуру каталогов и инструментов, которые позволяют извлекать, компилировать и устанавливать пакеты из исходного кода с помощью make-файлов. Обычно удобнее пользоваться менеджером **pkg**, но иногда предварительно скомпилированные пакеты недоступны.

Часто встречающиеся команды в пакетных менеджерах

Функциональность	**YUM**	**APT**
Установка пакета	yum install package_name	apt install package_name
Удаление пакета	yum remove package_name	apt remove package_name
Поиск пакета	yum search package_name	apt search package_name
Обновление	yum upgdate	apt update && apt dist-upgrade
Список установленных пакетов	yum list installed	apt list --installed
Поиск пакета, которому принадлежит файл	rpm -q --whatprovides /path/to/file	dpkg -S /path/to/file
Список файлов в пакете	rpm -ql package_name	dpkg -L package_name

Системы инициализации

Управление службами systemd (управление загрузкой и мониторингом активных служб и процессов): https://timeweb.cloud/tutorials/linux/kak-ispolzovat-systemctl-dlya-upravleniya-sluzhbami-systemd

Основы Ansible и git

Системы управления конфигурациями (SCM)

Системы управления конфигурацией (Configuration Management Systems, Software Configuration Management Systems) — программы и программные комплексы, позволяющие централизованно управлять конфигурацией множества разнообразных

разрозненных операционных систем и прикладного программного обеспечения, работающего в них.

Современные системы управления конфигурацией по сути стремятся к тому, чтобы в полной мере реализовать принцип Infrastructure-as-a-Code, в соответствеии с которым вся существующая IT-инфраструктура, машины, их конфигурация, связи между ними и так далее могут быть описаны одним или несколькими формальными файлами, а дальше это уже дело системы управления конфигурацией — воплотить описанную конфигурацию в жизнь.

Очень важно тут то, что состояние всей инфраструктуры остаётся обозримым и контроллируемым. Ручное выполнение операций на узлах минимизировано или сведено к нулю.

Использование Ansible

Ansible — система управления конфигурацией программного обеспечения.

В отличие от большинства других систем управления конфигурацией Ansible использует безагентную архитектуру. Агента нет, операции выполняются через SSH-подключение.

Достоинства

- Написан на Python и активно использует Python (для некоторых может это и не достоинство).
- Достаточно просто начать использование, не требует предварительной подготовки
- Использует стандартный и общедоступный протокол SSH (SSH-клиент и SSH-сервер) как основное средство коммуникации.
- Использование YAML для описания конфигурационных файлов.
- Не инсталлируется никакой дополнительный софт на управляемые машины, это хорошо в том числе с точки зрения безопасности, потому что не устанавливаются потенциально уязвимые программы.

Недостатки

- Требует SSH по умолчанию.
- Толком не поддерживает Windows.
- Довольно плохо масштабируется.
- Код модулей исполняется при импорте, поэтому чтобы красиво протестировать код в модулях, требуется немного магии. Если вы не любите лишнюю магию, это может быть недостатком.
- Не очень развитое комьюнити, велика роль одного человека.
- Нет консистентности между форматами входных, выходных и конфигурационных файлов.
- Для описания логики используется Jinja2, при росте сложности задачи нелинейно растёт сложность файлов.

Конфигурационные файлы и их приоритет

1. ANSIBLE_CONFIG
2. ansible.cfg

3. ~/.ansible.cfg
4. /etc/ansible/ansible.cfg

Сущности Ansible

1. Module — атомарное действие, которое можно выполнить с помощью команды ansible
2. Task — вызов модуля с параметрами
3. Playbook — несколько тасков или ролей
4. Role — структурированные плейбуки, состоящие из тасков

Повышение привилегий в Ansible

Для многих модулей Ansible необходимы права администратора на удаленной машине. Для этого в playbook может использоваться строка become: true (все модули автоматически исполняются с повышением привилегий).

(!) Повышение привилегий на удаленной машине запрашивается отдельно от авторизации по протоколу SSH, поэтому даже если авторизоваться по заранее настроенному ключу, для большинства модулей повышение привилегий все равно будет необходимо.

Опции для командной строки:

--ask-become-pass, -K – будет запрашиваться пароль для повышения привилегий (на всех хостах).

--become, -b – запуск модуля с повышением привилегий без пароля.

Подробное описание управления привилениями через Ansible в официальной документации:
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_privilege_escalation.html

Рекомендации по работе с Ansible

- Все конфигурационные файлы, плейбуки, роли нужно хранить в git
- Проектировать и писать роли, которые можно переиспользовать в разных проектах
- Все виртуальные машины описывать в отдельном inventory для каждого окружения
- Каждая роль должна содержать README.md (инструкцию) по использованию

Общий порядок работы с Ansible:

1. Создать inventory-файл (список хостов), который будет содержать группы ip-адресов, соответствующих тем или иным хостам, дать каждой группе уникальное имя, по которому можно к ней обращаться.
2. Создать playbook, который содержит указания, к каким группах хостов подключаться и с какими параметрами (необходимость повышения привилегий, пользователь, под которым нужно авторизоваться, роли, которые нужно запустить на выполнение).

3. Создать роли (roles), в которых будут описаны конкретные действия (модули), их описание и условия, при которых действия будут выполнены (например, на разных ОС установка ПО происходит с помощью различных пакетных менеджеров).
4. Запустить playbook (опционально – с указанием inventory-файла, если работа идет не с файлом по умолчанию), убедиться, что прописанные в playbook роли корректно выполняются.

(!) файл playbook может быть объединен с файлами ролей/сценариев (содержать помимо конфигурации модули и параметры их запуска), в некоторых примерах необходимо обращать на это внимание.

(!) в файлах *.yml важны отступы формируют структуру документа и иерархию элементов.

Например:

```
---
- name: Update Package on debian system
  hosts: all
  become: yes
  tasks:
  - name: Run the equivalent of "apt-get update" as a separate step
    apt:
      update_cache: yes
  - name: Install latest version of "openjdk-6-jdk"
    apt:
      name: openjdk-6-jdk
      state: latest
      install_recommends: no
```

Синтаксис YAML:

https://1cloud.ru/blog/yaml_for_beginners

Список основных модулей Ansible и примеры их использования:

https://habr.com/ru/companies/southbridge/articles/707130/

Структура Playbook:

https://habr.com/ru/companies/southbridge/articles/690614/

Примеры использования основных команд Ansible:

https://www.8host.com/blog/kak-rabotat-s-ansible-prostaya-i-udobnaya-shpargalka/

Создание пользовательских inventory-файлов:

https://www.8host.com/blog/sozdanie-fajla-inventarya-ansible/

Управление systemd с помощью Ansible (в примерах отступы yml-файлов отсутствуют, для корректной работы примеров отступы нужно добавить**):**

https://andreyex.ru/debian/kak-upravlyat-systemd-s-pomoshhyu-ansible/

Пример установки и настройки nginx с помощью Ansible:

https://www.dmosk.ru/instruktions.php?object=ansible-nginx-install

Ansible Galaxy

Ansible Galaxy - это репозиторий готовых плейбуков и ролей. Найти файлы, соответствующую задаче, а также документацию можно на портале galaxy.ansible.com. Установка выполняется командой **ansible-galaxy**, например:

```
ansible-galaxy install geerlingguy.apache
```

Данная команда создаст в каталоге пользователя папку ***.ansible/roles/geerlingguy.apache*** — в нее поместит файлы с описанием роли.

Готовые библиотеки можно использовать для выполнения задач или просто как шпаргалки.

Использование git

git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Для чего нужен git (система контроля версий):

- Совместная разработка нескольких людей/команд/компаний;
- Переносимость исходного кода между окружениями;
- Хранение истории изменений с описанием каждого из них;
- При использовании git становятся доступны все инструменты, ранее доступные только разработчикам (ветвления, слияния, code review, CI/CD).

Веб-сервисы основанные на системе контроля версий Git

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc.

Ссылка: https://github.com/

GitLab — веб-инструмент жизненного цикла DevOps с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики, системой отслеживания ошибок, CI/CD пайплайном и другими функциями. Код изначально был написан на Ruby, а некоторые его части были позже переписаны на Go.

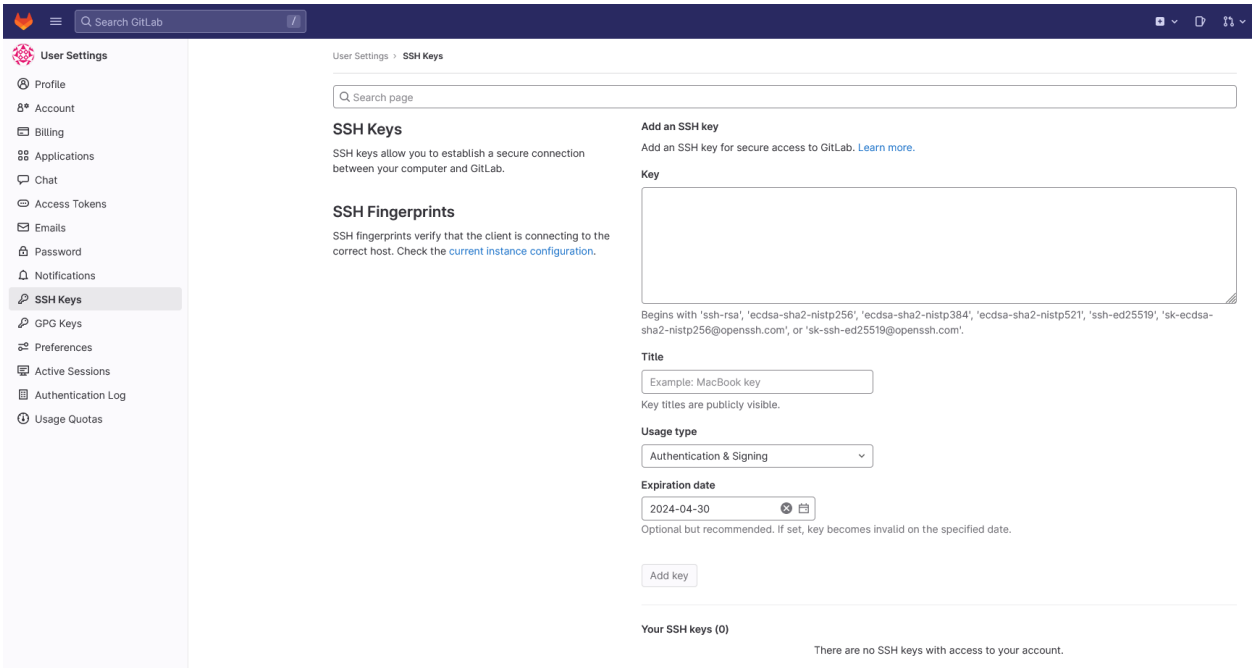
Ссылка: https://gitlab.com/

Bitbucket — веб-сервис для хостинга проектов и их совместной разработки, основанный на системах контроля версий Mercurial и Git.

Ссылка: https://bitbucket.org/

Чтобы начать работу с git посредством веб-сервисов, необходимо создать профиль пользователя одним из возможных способов и добавить SSH-ключ для авторизации.

Каждый сервис поддерживает авторизацию с помощью SSH, для этого нужно сгенерировать и добавить SSH-ключ в профиль пользователя.



Основные команды работы с git

Клонирование репозитория

```
$ git clone [repo_url]
```

Просмотр состояния файлов

```
$ git status
```

Отслеживание новых файлов

```
$ git add [file]
```

Коммитим изменения файла

```
$ git commit –m "Сообщение коммита"
```

```
$ git status
```

Отправка изменений

```
$ git push [remote-name]
```

Пример:

```
$ git push new_repo master
```

(!) В рамках лабораторной работы допустимо использовать графические оболочки git, например git-cola: https://git-cola.github.io/ (также можно установить через менеджер пакетов графической оболочки Linux Mint).

Подробный пример по созданию репозитория git и работе с ним: https://www.atlassian.com/ru/git/tutorials/setting-up-a-repository

Задание на лабораторную работу

Цель: Изучить основные инструменты для работы с файловой системой. Научиться управлять стандартными потоками Linux. Научиться устанавливать, настраивать Ansible и подготавливать к работе playbooks. Освоить работу с системой контроля версий Git.

1. **Создать LVM с новым диском, подключенным к ВМ**. Создать структуру как минимум с одним Physical Volume, Volume Group и Logical Volume. Создать на нем ФС ext4, которая будет монтироваться в директорию /mnt/storage при загрузке ВМ.

2. Описать все эти действия в скрипте, снабженном комментариями.

3. **Форматирование вывода.** Вывести на экран название PV и соответствующих им VG (дополнительно: в формате “PV = VG”), используя команду вывода из состава LVM.

4. **Настроить второй сетевой адаптер виртуальной машины** для организации внутренней сети между виртуальными машинами.

5. **Клонировать виртуальную машину с Linux.** Новая виртуальная машина будет играть роль хоста, на котором будет проводиться удаленная конфигурация с помощью Ansible. Дать машине при клонировании имя с суффиксом *«_host»*.

6. **Повторить процесс настройки SSH-подключения** от основной виртуальной машины к новой, аналогично пп. 4-6 первой лабораторной.

7. **Написать скрипт**, использующий все стандартные потоки (ввод, вывод, ошибки) и обрабатывающий код возврата (см. примеры в методических указаниях).

8. **Установить на основной виртуальной машине git и ansible** с помощью пакетного менеджера.

9. **Установить с помощью Ansible на клоне виртуальной машины nginx и интерпретатор PHP**. Обе виртуальные машины должны быть включены одновременно. Использовать для установки только команды для удаленной установки Ansible на основной машине. В результате ПО должно быть установлено на клоне (*«_host»*).

10. **Написать playbook,** который будет содержать установку и настройку поведения сервиса nginx: должен включаться при старте ОС. Включить сервис.

11. **Зарегистрироваться на одном из веб-сервисов git**.

12. **Сгенерировать ssh-ключ или использовать существующий**.

13. **Добавить ssh-ключ в профиль своего пользователя git**.

14. **Создать новый репозиторий с помощью git**, сохранить в нем созданные файлы Ansible (playbook, inventory и т.д.), сделать push на удаленный веб-сервер из п. 11. Показать, что файлы успешно загружены.

Результат: создали новую структуру томов с помощью LVM. Настроили сетевое соединение между двумя виртуальными машинами. Установили, настроили Ansible и подготовили playbook. Изучили систему контроля версий git.

ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.
2. Цели выполняемой лабораторной работы.
3. Задание на лабораторную работу.
4. Описание процесса выполнения работы: для каждого действия, производимого в командной строке, в отчет следует включить:
   - — краткое описание действия;
   - — вводимая команда или команды;
   - — реакция системы на ввод команд (если объем выводимых данных превышает несколько строк, всю информацию включать в отчет не следует).
5. Выводы.