MyLittleWiki Documentation

Выпуск latest

дек. 22, 2020

Оглавление

Глава 1

Железо

1.1 Обновление BIOS на серверах Supermicro

Примечание: Самый простой способ обновить прошивку *BIOS* на серверах *Supermicro*, сделать это через веб-интерфейс IPMI сервера. Но для этого нужно приобрести ключ. Описанный ниже способ расскаждет как обновить *BIOS* не имея ключа.

Скачиваем архив с обновлением *BIOS* для нужной материнской платы с сайта Supermicro.

Можно воспользоваться поиском, или посмотреть на странице.

Скачиваем образ с сайта Hiren Boot CD.

Распаковываем архив с обновлением прошивки в одтельную папку.

Открываем образ *Hiren Boot CD* в любом редакторе iso-образов и добавляем в корень диска папку с файлами для обновления *BIOS*. Сохраняем образ.

Монтируем образ на виртуальный привод *IPMI* и загружаемся *DOS* согласно инструкции.

Копируем файлы все из директории содержащей новую прошивку *BIOS* в корень RAM-диска *R*. Обычно директория с файлами находится на диске `C:\`.

К примеру, директория называется *bios*:

```
R:\$> C:
C:\$> CD bios\
C:\$> COPY *.* R:\*.*
C:\$> R:
```

Запускаем обновление прошивки. Обычно это производится скриптом `FLASH.BAT`, но более точную информацию по процедуре прошивки можно прочесть в файле `Readme.txt` входящего в состав файлов архива обновлений.

Пример обновления:

```
R:\$> FLASH.BAT X11SSE8.516
```

Ожидаем окончания процесса обновления прошивки.

Внимание: После обновления прошивки, сервер необходимо выключить по питанию минимум на 10 секунд.

После включения зайти в *BIOS* и восстановить необходимые настройки.

Глава 2

Сеть

2.1 Настройка DNS сервера Bind на Ubuntu 16.04

Перед началом установки мы рекомендуем вам обновить информацию о пакетах, содержащихся в репозиториях:

```
sudo apt-get update
```

2.1.1 Установка *bind9*

BIND *(Berkeley Internet Name Domain)* является реализацией протокола *Domain Name System (DNS)* с открытым исходным кодом *(Open-source)*, и обеспечивает реализацию основных компонентов системы доменных имен.

Вы можете установить *Bind9* используя команду:

```
sudo apt-get install bind9 dnsutils
```

2.1.2 Настройка *Bind* для протокола *IPv4*

После установки необходимо отредактировать файл `named.conf.local`

```
sudo nano /etc/bind/named.conf.local
```

В этот файл добавьте информацию о ваших зонах. В качестве примера будем использовать доменное имя *dnstest.root.lu* и подсеть 94.242.218.160/28 (94.242.218.162 основной *IP сервера*, на котором настраивается *DNS*).

```
zone "dnstest.root.lu" {
  type master;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  file "/etc/bind/zones/db.dnstest.root.lu";
};

zone "218.242.94.in-addr.arpa" {
  type master;
  file "/etc/bind/zones/db.94";
};
```

Нажмите `Ctrl+X` чтобы сохранить файл и выйти из редактора.

Теперь необходимо отредактировать файл опций *Bind9*.

```
sudo nano /etc/bind/named.conf.options
```

Укажите `forwarders` - адреса вышестоящих *DNS серверов*, куда ваш *DNS сервер* будет перенаправлять запросы которые не сможет обработать самостоятельно.

```
forwarders {
   8.8.8.8; # Google Public DNS IPv4 address
   8.8.4.4; # Google Public DNS IPv4 address
};
```

Далее, добавляем файл зоны прямого просмотра:

```
sudo mkdir /etc/bind/zones
sudo nano /etc/bind/zones/db.dnstest.root.lu
```

Вносим в файл информацию о нашей зоне, и адреса/имена хостов, которые будут определяться нашим *DNS сервером*.

```
$TTL 604800
@ IN SOA ns.dnstest.root.lu. admin.dnstest.root.lu. (
   2 ; Serial
   604800 ; Refresh
   86400 ; Retry
   2419200 ; Expire
   604800 ) ; Negative Cache TTL

;dnsserver
@   IN NS ns.dnstest.root.lu.
@   IN A  94.242.218.162
ns  IN A  94.242.218.162

;clients
subdomain1 IN A  94.242.218.163
subdomain2 IN A  94.242.218.164
subdomain3 IN A  94.242.218.165
```

Теперь создадим файл обратной зоны просмотра (*reverse DNS zone*).

```
sudo nano /etc/bind/zones/db.94
```

Добавим в файл следующую информацию:

```
$TTL 604800
@ IN SOA ns.dnstest.root.lu. admin.dnstest.root.lu. (
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    1 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@   IN NS  ns.dnstest.root.lu.
162 IN PTR dnstest.root.lu.

163 IN PTR subdomain1.dnstest.root.lu.
164 IN PTR subdomain2.dnstest.root.lu.
165 IN PTR subdomain3.dnstest.root.lu.
```

Теперь проверим правильность синтаксиса конфигурационных файлов *BIND*.

```
named-checkconf
```

Если конфигурационные файлы не содержат ошибок, вывод этой программы будет пуст.

Далее, необходимо перезапустить сервис *Bind*.

```
sudo service bind9 restart
```

И проверить статус *Bind* на наличие ошибок.

```
sudo service bind9 status
```

```
 bind9.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/bind9.service; enabled; vendor preset: enabled)
   Drop-In: /run/systemd/generator/bind9.service.d
            └─50-insserv.conf-$named.conf
   Active: active (running) since Sat 2016-04-23 21:00:17 CEST; 2min 52s ago
```

Если в результате выполнения вы увидите ошибки, ещё раз перепроверьте правильность данных в конфигурационных файлах *Bind* и в директории `/etc/bind/zones/`

Теперь проверим работу вашего *DNS сервера*. Для начала изменим содержимое файла `resolv.conf`.

```
sudo nano /etc/resolvconf/resolv.conf.d/head
```

Вставьте в файл следующие строки с данными вашего *DNS сервера* и сохраните файл.

```
search dnstest.root.lu
nameserver 94.242.218.162
```

Выполните команду `resolvconf` чтобы сгенерировать новый файл `resolv.conf`.

```
sudo resolvconf -u
```

И наконец, протестируем ваш *DNS* при помощи одной из двух приведённых ниже команд.

Резолв зоны `dnstest.root.lu`:

```
dig @94.242.218.162 dnstest.root.lu
```

```
;; ANSWER SECTION:
dnstest.root.lu.     604800  IN      A       94.242.218.162

;; AUTHORITY SECTION:
dnstest.root.lu.     604800  IN      NS      ns.dnstest.root.lu.

;; ADDITIONAL SECTION:
ns.dnstest.root.lu.  604800  IN      A       94.242.218.162
```

Обратный резольв IPv4:

```
dig @94.242.218.162 -x 94.242.218.162
```

```
;; ANSWER SECTION:
162.218.242.94.in-addr.arpa. 604800 IN  PTR     dnstest.root.lu.

;; AUTHORITY SECTION:
218.242.94.in-addr.arpa. 604800     IN  NS      ns.dnstest.root.lu.
```

Если вы получили подобный результат, значит ваш *DNS сервер* сконфигурирован правильно.

2.1.3 Настройка *Bind* для *IPv6*

В качестве примера к серверу прикреплена *IPv6* подсеть 2a01:608:ffff:a02b::/64. Основной адрес, настроеный на интерфейсе сервера - 2a01:608:ffff:a02b::2

Сначала в файле `/etc/bind/named.conf.options` подправим секцию `forwarders`, пропишем публичный *IPv6 DNS сервер* компании **Google**.

```
sudo nano /etc/bind/named.conf.options
```

Файл `/named.conf.options` должен иметь следующий вид:

```
options {
  directory "/var/cache/bind";

  // If there is a firewall between you and nameservers you want
  // to talk to, you may need to fix the firewall to allow multiple
  // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

  // If your ISP provided one or more IP addresses for stable
  // nameservers, you probably want to use them as forwarders.
  // Uncomment the following block, and insert the addresses replacing
  // the all-0's placeholder.

  forwarders {
        8.8.8.8; # Google Public DNS IPv4 addresses
        8.8.4.4; # Google Public DNS IPv4 addresses
        2001:4860:4860::8888; # Google Public DNS IPv6 address
        2001:4860:4860::8844; # Google Public DNS IPv6 address
  };

  //========================================================================
  // If BIND logs error messages about the root key being expired,
  // you will need to update your keys.  See https://www.isc.org/bind-keys
  //========================================================================
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  dnssec-validation auto;

  auth-nxdomain no;    # conform to RFC1035
  listen-on-v6 { any; };
};
```

Далее, добавьте следующую информацию в файл `/etc/bind/zones/db.dnstest.root.lu`

```
;dnsserver
@   IN AAAA 2a01:608:ffff:a02b::2
ns  IN AAAA 2a01:608:ffff:a02b::2

;clients
subdomain1 IN AAAA 2a01:608:ffff:a02b::3
subdomain2 IN AAAA 2a01:608:ffff:a02b::4
subdomain3 IN AAAA 2a01:608:ffff:a02b::5
```

Для настройки обратного просмотра нужно вписать дополнительный диапазон адресов в `/etc/bind/named.conf.local`. В конце файла добавьте строки, содержащие объявление диапазона (*zone*), и не забудьте что каждый шестнадцатиричный блок адреса нужно дополнять нулём (если необходимо), для того чтобы он содержал 4 символа:

```
sudo nano /etc/bind/named.conf.local
```

```
zone "b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa" {
         type master;
         notify no;
         file "/etc/bind/zones/db.b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa";
};
```

Теперь создадим новый файл обратной зоны для *IPv6*:

```
sudo nano /etc/bind/zones/db.b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa
```

Заполним его следующим содержимым:

```
$ORIGIN b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa.
$TTL 1d
@  IN  SOA  dnstest.root.lu. admin.dnstest.root.lu. (
        2014011501      ;Serial
        86400           ;Refresh
        7200            ;Retry
        2592000         ;Expire
        172800          ;Minimum TTL
)
    IN  NS  ns.dnstest.root.lu.

; 2a01:608:ffff:a02b::/64
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0  IN  PTR  ns.dnstest.root.lu.

3.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0  IN  PTR  subdomain1.dnstest.root.lu.
4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0  IN  PTR  subdomain2.dnstest.root.lu.
5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0  IN  PTR  subdomain3.dnstest.root.lu.
```

В PTR записи необходимо указать только адрес хоста (без адреса подсети). Каждый блок адреса должен быть дополнен нулями.

Подсказка: К примеру адрес *DNS сервера* (2a01:608:ffff:a02b::2) имеет адрес подсети „2a01:608:ffff:a02b“ и адрес хоста „2“. Адрес хоста преобразован в „0000:0000:0000:0002“ и записан в обратном порядке, с разделителями в виде точек.

После этого не забудьте перезапустить сервис *Bind* и проверить его статус.

```
sudo service bind9 restart
sudo service bind9 status
```

```
 bind9.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/bind9.service; enabled; vendor preset: enabled)
   Drop-In: /run/systemd/generator/bind9.service.d
            └─50-insserv.conf-$named.conf
   Active: active (running) since Sat 2016-04-23 23:38:16 CEST; 2min 31s ago
```

Итак, вывод статуса без ошибок. Теперь можно проверить правильность конфигурации *DNS/rDNS* зоны для *IPv6*.

Резолв зоны `dnstest.root.lu`:

```
dig @94.242.218.162 dnstest.root.lu
```

```
;; ANSWER SECTION:
dnstest.root.lu.        604800  IN      A       94.242.218.162

;; AUTHORITY SECTION:
dnstest.root.lu.        604800  IN      NS      ns.dnstest.root.lu.

;; ADDITIONAL SECTION:
ns.dnstest.root.lu.     604800  IN      A       94.242.218.162
ns.dnstest.root.lu.     604800  IN      AAAA    2a01:608:ffff:a02b::2
```

Обратный резольв *IPv6*:

```
dig @94.242.218.162 -x 2a01:608:ffff:a02b::2
```

```
;; ANSWER SECTION:
2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa. 86400 IN PTR ns.dnstest.
↪root.lu.

;; AUTHORITY SECTION:
b.2.0.a.f.f.f.f.8.0.6.0.1.0.a.2.ip6.arpa. 86400 IN NS ns.dnstest.root.lu.
```

Ваш *DNS сервер* теперь готов к работе!

Глава 3

Windows

3.1 Добавление драйверов в boot.wim

Сначала необходимо открыть *cmd* (командную строку Windows) от имени *Администратора*.

Получение информации о содержимом образа `boot.wim`:

```
C:\Users\Administrator> dism /get-imageinfo /imagefile:C:\boot.wim

Deployment Image Servicing and Management tool
Version: 6.3.9600.16384

Details for image : C:\boot.wim

Index : 1
Name : Microsoft Windows PE (x64)
Description : Microsoft Windows PE (x64)
Size : 1,321,549,982 bytes

Index : 2
Name : Microsoft Windows Setup
Description : Microsoft Windows Setup
Size : 1,417,514,940 bytes

The operation completed successfully.
```

Смонтировать образ в директорию `C:\mnt`:

```
C:\Users\Administrator> dism /mount-image /imagefile:C:\boot.wim /index:1 /mountdir:C:\mnt

Deployment Image Servicing and Management tool
Version: 6.3.9600.16384

Mounting image
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[==========================100.0%==========================]
The operation completed successfully.
```

Добавление в распакованный образ драйверов из директории `C:\drivers`:

```
dism /image:C:\mnt /add-driver /driver:C:\drivers /recurse /forceunsigned

Deployment Image Servicing and Management tool
Version: 6.3.9600.16384

Image Version: 6.3.9600.16384

Searching for driver packages to install...
Found 2 driver package(s) to install.
Installing 1 of 2 - C:\drivers\3wManage.inf: The driver package was successfully installed.
Installing 2 of 2 - C:\drivers\oemsetup.inf: The driver package was successfully installed.
The operation completed successfully.
```

- `/forceunsigned` - добавление неподписанных драйверов
- `/recurse` - рекурсивное добавление всех драйверов в директории

Если же необходимо добавить конкретный драйвер, то нужно указать точно `inf-файл`. Пример:

```
dism /image:C:\mnt /add-driver /driver:C:\drivers\oemsetup.inf /forceunsigned
```

Проверка списка добавленных драйверов:

```
C:\Users\Administrator> dism /image:C:\mnt /get-drivers

Deployment Image Servicing and Management tool
Version: 6.3.9600.16384

Image Version: 6.3.9600.16384

Obtaining list of 3rd party drivers from the driver store...

Driver packages listing:

Published Name : oem0.inf
Original File Name : 3wmanage.inf
Inbox : No
Class Name : System
Provider Name : AMCC
Date : 3/24/2009
Version : 3.0.4.70

Published Name : oem1.inf
Original File Name : oemsetup.inf
Inbox : No
Class Name : SCSIAdapter
Provider Name : AMCC
Date : 3/24/2009
Version : 3.0.4.70

The operation completed successfully.
```

Сохранение и размонтирование образа:

```
C:\Users\Administrator> dism /unmount-image /mountdir:C:\mnt /commit

Deployment Image Servicing and Management tool
Version: 6.3.9600.16384

Saving image
[==========================100.0%==========================]
Unmounting image
[==========================100.0%==========================]
The operation completed successfully.
```

Остаётся только заменить загрузочный образ на WDS на созданный.

Глава 4

Linux

4.1 Debian Linux

4.1.1 Интеграция non-free драйверов в *initrd.gz*

1. Создать резервную копию `initrd.gz`

```
$ [ -f initrd.gz.orig ] || \
  cp -p initrd.gz initrd.gz.orig
```

2. Скачать архив со свежимы *deb*-пакетами драйверов

```
$ [ -f firmware.cpio.gz ] || \
  wget http://cdimage.debian.org/cdimage/unofficial/non-free/firmware/stable/current/firmware.cpio.
↪gz
```

3. Добавить драйвера в загрузочный образ `initrd.gz`

```
$ cat initrd.gz.orig firmware.cpio.gz > initrd.gz
```

4.2 Deepin Linux

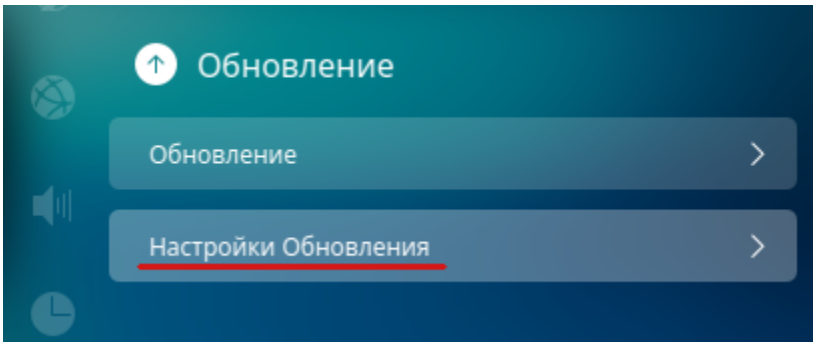
4.2.1 Изменение зеркала обновления Deepin linux

Как правило после установки Deepin linux зеркалом обновления пакетов по-умолчанию назначается основной репозиторий, находящийся в Китае. И хотя в последней версии (на момент написания это версия 15.8) появилась опция автоматического подключения к самому быстрому зеркалу, но на практике это не всегда происходит и обновления скачиваются с основного репозитория с невысокой скоростью загрузки.

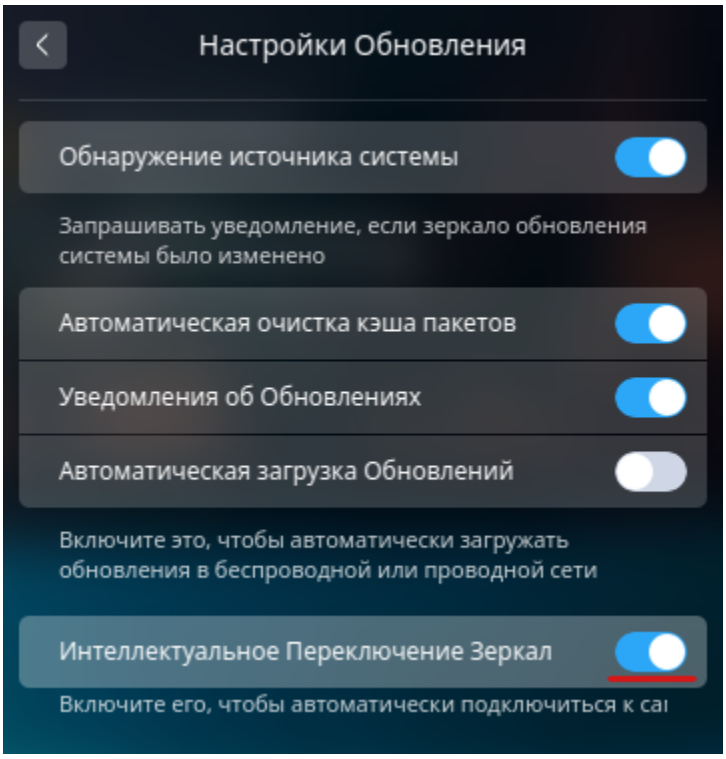
Есть два варианта сменить зеркало обновления Deepin linux:

Изменение зеракла обновлений через *Центр управления*

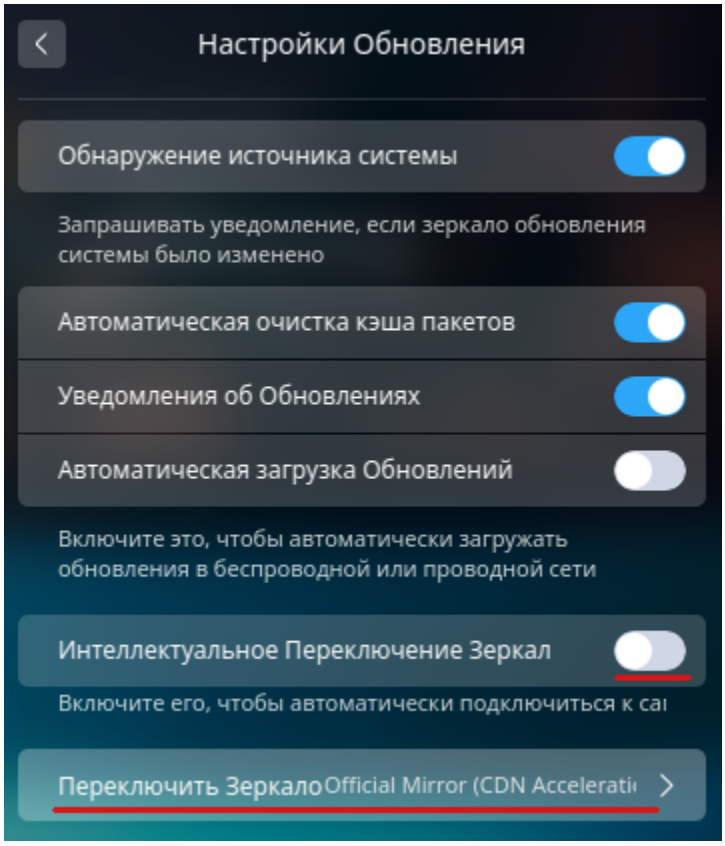
1. Необходимо зайти в раздел *Центра управления* -> *Обновление* -> *Настройки обновления*



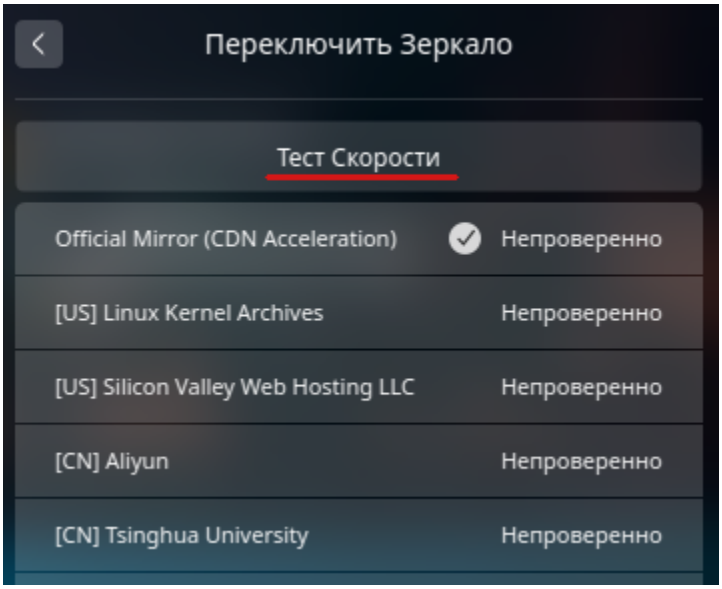
2. Отключить *Интеллектуальное обновление зеркал*



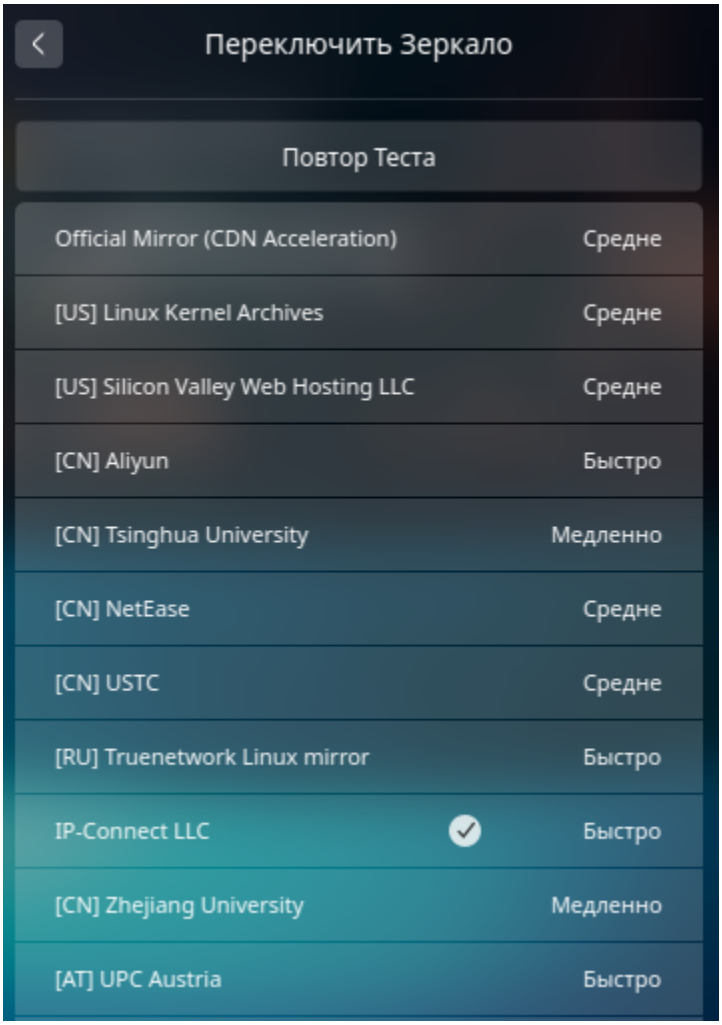
3. После отключения *Интеллектуального обновления зеркал* ниже появится новый пункт *Переключить зеркало*



4. Заходим в раздел *Переключить зеркало* и нажимаем кнопку *Тест скорости*, чтобы определить наоболее подходящее зеркало.



5. После теста, выбираем одно из зеркал у которых будет написано *быстро*



На этом настройка нового зеркала обновлений закончена. Остаётся только обновить кэш пакетов.

```
sudo apt update
```

Изменение зеркала обновлений непосредственно в файле `sources.list`

Все действия необходимо выполнять от имени `root` или через команду `sudo`

1. Сохраним оригинал файла `sources.list`

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.original
```

2. Открываем файл `sources.list` в любимом текстовом редакторе:

```
sudo nano /etc/apt/sources.list
```

и добавляем в него новое зеркало. Старые записи коментируем.

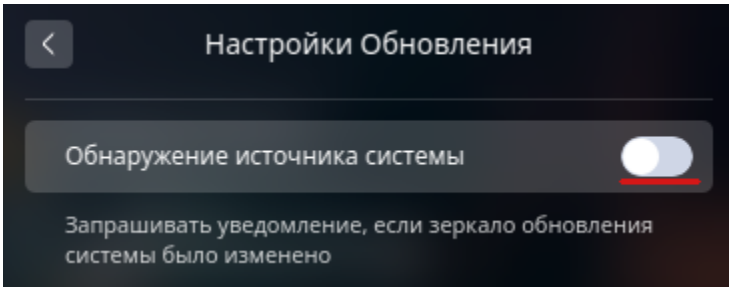
```
## Generated by deepin-installer
# deb [by-hash=force] http://packages.deepin.com/deepin panda main contrib non-free
# deb-src http://packages.deepin.com/deepin panda main contrib non-free
deb http://mirror.inode.at/data/deepin panda main contrib non-free
```

3. Сохраняем файл и обновляем кэш пакетов.

```
sudo apt update
```

Новое зеркало обновлений настроено.

Примечание: После ручного изменения файла `sources.list` может выдаваться предупреждение что основной репозиторий изменён с предложением переключиться на репозиторий по-умолчанию. Чтобы такое предупреждение не появлялось, необходимо в *Центр управления -> Обновление* отключить *Обнаружение источника системы*



4.3 Терминал Linux

4.3.1 FAQ

Как узнать дату установки Ubuntu?

Для этого нужно в терминал ввести команду:

```
ls -lct /etc | tail -1 | awk '{print $6,$7,$8}'
```

Пример результата выполнения:

```
Feb 28 16:35
```

Как распаковать и собрать заново образ загрузки initrd (initramfs)?

Создать временную директорию, в которую распакуем initrd:

```
mkdir /tmp/initramfscd /tmp/initramfs
```

Распаковать:

```
gunzip -c -9 /boot/initrd.img-2.6.18-1-686 | cpio -i -d -H newc --no-absolute-filenames
```

Запаковать образ:

```
cd /tmp/initramfsfind . | cpio -o -H newc | gzip -9 > /boot/initrd.img-2.6.18-1-686
```

Как посмотреть версию SATA?

существует уже целых три версии интерфейса SATA и каждая версия имеет свое ограничение по скорости:

- SATA revision 1.0 – 1.5 Гбит/с – 150 Мбайт/с
- SATA revision 2.0 – 3 Гбит/с – 300 Мбайт/с
- SATA revision 3.0 – 6 Гбит/с – 600 Мбайт/с

Как же узнать предельную скорость, которую поддерживает ваша материнская плата? Пользователи Windows могут вызвать платного консультанта, который вскроет корпус, заглянет вовнутрь и прочитает соответствующие надписи на микросхемах. Пользователи Linux могут узнать версию SATA, написав в терминале всего одну строчку:

```
sudo hdparm -I /dev/sda | grep -E "Model | speed"
```

В результате на экране появится нечто подобное:

```
Model Number:       ST1000LM035-1RK172
    *        Gen1 signaling speed (1.5Gb/s)
    *        Gen2 signaling speed (3.0Gb/s)
    *        Gen3 signaling speed (6.0Gb/s)
```

4.3.2 Использование *grep*

Поиск текста в файлах

В первом примере мы будем искать пользователя User в файле паролей Linux. Чтобы выполнить поиск текста grep в файле /etc/passwd введите следующую команду:

```
$ grep User /etc/passwd
```

В результате вы получите что-то вроде этого, если, конечно, существует такой пользователь:

```
User:x:1000:1000:User,,,:/home/User:/bin/bash
```

А теперь не будем учитывать регистр во время поиска. Тогда комбинации ABC, abc и Abc с точки зрения программы будут одинаковы:

```
$ grep -i "user" /etc/passwd
```

Вывести несколько строк

Например, мы хотим выбрать все ошибки из лог-файла, но знаем, что в следующей строчке после ошибки может содержаться полезная информация, тогда с помощью grep отобразим несколько строк. Ошибки будем искать в Xorg.log по шаблону «EE»:

```
$ grep -A4 "EE" /var/log/xorg.0.log
```

Выведет строку с вхождением и 4 строчки после неё:

```
$ grep -B4 "EE" /var/log/xorg.0.log
```

Выведет целевую строку и 4 строчки до неё:

```
$ grep -C2 "EE" /var/log/xorg.0.log
```

Выведет по две строки с верху и снизу от вхождения.

Регулярные выражения в *grep*

Регулярные выражения grep - очень мощный инструмент в разы расширяющий возможности поиска текста в файлах. Для активации этого режима используйте опцию -e. Рассмотрим несколько примеров:

Поиск вхождения в начале строки с помощью спецсимвола «^», например, выведем все сообщения за ноябрь:

```
$ grep "^Nov 10" messages.1

Nov 10 01:12:55 gs123 ntpd[2241]: time reset +0.177479 s
Nov 10 01:17:17 gs123 ntpd[2241]: synchronized to LOCAL(0), stratum 10
```

Поиск в конце строки - спецсимвол «$»:

```
$ grep "terminating.$" messages

Jul 12 17:01:09 cloneme kernel: Kernel log daemon terminating.
Oct 28 06:29:54 cloneme kernel: Kernel log daemon terminating.
```

Найдём все строки, которые содержат цифры:

```
$ grep "[0-9]" /var/log/Xorg.0.log
```

Рекурсивный поиск фрагмента в файлах.

```
$ grep -r "mydomain.com" /etc/apache2/
```

Кавычки позволяют искать полное соотвествие с заданным шаблоном. Пример:

```
$ grep -r "zendsite" /etc/apache2/
/etc/apache2/vhosts.d/zendsite_vhost.conf: ServerName zendsite.localhost
/etc/apache2/vhosts.d/zendsite_vhost.conf: DocumentRoot /var/www/localhost/htdocs/zendsite
/etc/apache2/vhosts.d/zendsite_vhost.conf: <Directory /var/www/localhost/htdocs/zendsite>
```

Здесь перед найденной строкой указано имя файла, в котором она была найдена. Добавление параметра -n добавляет к выводу номер строки в файле, где найден заданный шаблон поиска. Вывод имени файла легко отключить с помощью опции -h:

```
$ grep -h -r "zendsite" /etc/apache2/

ServerName zendsite.localhost
DocumentRoot /var/www/localhost/htdocs/zendsite
<Directory /var/www/localhost/htdocs/zendsite>
```

Это будет искать только те файлы, у которых есть.c или.h расширения:

```
$ grep --include=\*.{c,h} -rnw '/path/to/somewhere/' -e "pattern"
```

Это исключает поиск всех файлов, заканчивающихся расширением.o:

```
$ grep --exclude=*.o -rnw '/path/to/somewhere/' -e "pattern"
```

Для каталогов можно исключить конкретный каталог через параметр –exclude-dir. Например, это исключает dir1/, dir2/dir2 и все из них, соответствующие *.dst/:

```
$ grep --exclude-dir={dir1,dir2,*.dst} -rnw '/path/to/somewhere/' -e "pattern"
```

Поиск слов в *grep*

Когда вы ищете строку abc, grep будет выводить также kbabc, abc123, aafrabc32 и тому подобные комбинации. Вы можете заставить утилиту искать по содержимому файлов в Linux только те строки, которые выключают искомые слова с помощью опции -w:

```
$ grep -w "abc" имя_файла
```

Поиск двух слов

Можно искать по содержимому файла не одно слово, а два сразу:

```
$ egrep -w 'word1|word2' /path/to/file
```

Количество вхождений строки

Утилита grep может сообщить, сколько раз определённая строка была найдена в каждом файле. Для этого используется опция -c (счетчик):

```
$ grep -c 'word' /path/to/file
```

Инвертированный поиск в grep

Команда grep Linux может быть использована для поиска строк в файле, которые не содержат указанное слово. Например, вывести только те строки, которые не содержат слово пар:

```
$ grep -v пар /path/to/file
```

Вывод имени файла

Вы можете указать grep выводить только имя файла, в котором было найдено заданное слово с помощью опции -l. Например, следующая команда выведет все имена файлов, при поиске по содержимому которых было обнаружено вхождение primary:

```
$ grep -l 'primary' *.c
```

Цветной вывод в *grep*

Также вы можете заставить программу выделять другим цветом вхождения в выводе:

```
$ grep --color root /etc/passwd
```

4.3.3 Подсветка синтаксиса в *nano*

Настройки подсветки сиснтаксиса в консольном текстовом редакторе *nano* по-умолчанию подключаются в конфигурационном файле `/etc/nanorc`

```
$ cat /etc/nanorc
...
## To include all existing syntax definitions, you can do:
include "/usr/share/nano/*.nanorc"
```

Из этой записи видно, что конфиги подсветки синтаксиса нужно искать по пути `/usr/share/nano/`

```
$ ls -l /usr/share/nano
-rw-r--r-- 1 root root  882 дек 18 01:19 asm.nanorc
-rw-r--r-- 1 root root  555 дек 18 01:19 autoconf.nanorc
-rw-r--r-- 1 root root 1350 дек 18 01:19 awk.nanorc
-rw-r--r-- 1 root root  709 дек 18 01:19 changelog.nanorc
-rw-r--r-- 1 root root  825 дек 18 01:19 cmake.nanorc
...
```

Но если открыть *nano* от юзера, то подсветка синтаксиса не работает.

Включение подсветки синтаксиса для юзера.

Чтобы её включить необходимо создать файл `.nanorc`

```
$ touch ~/.nanorc
```

И добавить в него инклюды на файлы подсветки синтаксиса.

```
$ find /usr/share/nano/ -iname "*.nanorc" -exec echo include {} \; >> ~/.nanorc
```

Как можно заметить, в списке отсутсвуют конфиги для синтаксиса *conf* и *yaml* файлов.

Добавление подсветки синтаксиса для *conf* файлов.

Открываем новый файла

```
$ sudo nano /usr/share/nano/conf.nanorc
```

И добавляем в него следующее содержимое

```
# config file highlighting

syntax "conf" "(\.
↪(conf|config|cfg|cnf|rc|lst|list|defs|ini|desktop|mime|types|preset|cache|seat|service|htaccess)
↪$|(^|/
↪)(\w*crontab|mirrorlist|group|hosts|passwd|rpc|netconfig|shadow|fstab|inittab|inputrc|protocols|sudoers)
↪$|conf.d/|.config/)"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# default text
color magenta "^.*$"
# special values
icolor brightblue "(^|\s|=)(default|true|false|on|off|yes|no)(\s|$)"
# keys
icolor cyan "^\s*(set\s+)?[A-Z0-9_\/\.\%\@+-]+\s*([:]|\>)"
# commands
color blue "^\s*set\s+\<"
# punctuation
color blue "[.]"
# numbers
color red "(^|\s|[[/:|<>(){}=,]|\])[-+]?[0-9](\.?[0-9])*%?($|\>)"
# keys
icolor cyan "^\s*(\$if )?([A-Z0-9_\/\.\%\@+-]|\s)+="
# punctuation
color blue "/"
color brightwhite "(\]|[()<>[{},;:=])"
color brightwhite "(^|\[|\{|\:)\s*-(\s|$)"
# section headings
icolor brightyellow "^\s*(\[([A-Z0-9_\.-]|\s)+\])+\s*$"
color brightcyan "^\s*((Sub)?Section\s*(=|\>)|End(Sub)?Section\s*$)"
color brightcyan "^\s*\$(end)?if(\s|$)"
# URLs
icolor green "\b(([A-Z]+://|www[.])[A-Z0-9/:#?&$=_\.\-]+)(\b|$| )"
# XML-like tags
icolor brightcyan "</?\w+((\s*\w+\s*=)?\s*("[^"]*"|'[^']*'|!?[A-Z0-9_:/]))*(\s*/)?>"
# strings
color yellow "\"(\\.|[^"])*\"" "'(\\.|[^'])*'"
# comments
color white "#.*$"
color blue "^\s*##.*$"
color white "^;.*$"
color white start="<!--" end="-->"
```

После сохранения файла его необходимо добавить в `.nanorc`

```
$ echo include "/usr/share/nano/conf.nanorc" >> ~/.nanorc
```

Добавление подсветки синтаксиса для *yaml* файлов.

Открываем новый файла

```
$ sudo nano /usr/share/nano/yaml.nanorc
```

И добавляем в него следующее содержимое

```
# Supports `YAML` files
syntax "YAML" "\.ya?ml$"
header "^(---|===)" "%YAML"

## Keys
color magenta "^\s*[\$A-Za-z0-9_-]+\:"
color brightmagenta "^\s*@[\$A-Za-z0-9_-]+\:"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
## Values
color white ":\s.+$"
## Booleans
icolor brightcyan " (y|yes|n|no|true|false|on|off)$"
## Numbers
color brightred " [[:digit:]]+(\.[[:digit:]]+)?"
## Arrays
color red "\[" "\]" ":\s+[|>]" "^\s*- "
## Reserved
color green "(^| )!!(binary|bool|float|int|map|null|omap|seq|set|str) "

## Comments
color brightwhite "#.*$"

## Errors
color ,red ":\w.+$"
color ,red ":'.+$"
color ,red ":".+$"
color ,red "\s+$"

## Non closed quote
color ,red "['\"][^['\"]]*$"

## Closed quotes
color yellow "['\"].*['\"]"

## Equal sign
color brightgreen ":( |$)"
```

После сохранения файла его необходимо добавить в `.nanorc`

```
$ echo include "/usr/share/nano/yaml.nanorc" >> ~/.nanorc
```

4.3.4 Примеры использования find

Найти в текущем каталоге (рекурсивно) файлы старее 25-ти дней

```
find -type f -mtime +25
```

Найти в текущем каталоге (рекурсивно) файлы старее 25-ти дней и удалить

```
find -type f -mtime +25 -exec rm -rf {} \;
```

Найти (рекурсивно) файлы старее 25-ти дней в директории `/home/user`

```
find /home/user -type f -mtime +25
```

Удаление файлов старше 10-ти дней

```
find /dir -atime +10 -delete
```

В директории `/dir` найти и удалить все файлы с расширением `jpg` старше 10-ти дней

```
find /dir -name '*.jpg' -mtime +10 -exec rm -f {} \;
```

Ключи:

-name — искать по имени файла, при использовании подстановочных образцов параметр заключается в кавычки.

-type — тип искомого: *f=файл, d=каталог, l=ссылка (link).*

-user — владелец: имя пользователя или *UID.*

-group — владелец: группа пользователя или *GID.*

-perm — указываются права доступа.

-size — размер: указывается в 512-байтных блоках или байтах (признак байтов — символ «c» за числом).

-atime — время последнего обращения к файлу.

-ctime — время последнего изменения владельца или прав доступа к файлу.

-mtime — время последнего изменения файла.

-newer другойфайл — искать файлы созданные позже, чем другойфайл.

-delete — удалять найденные файлы.

-ls — генерирует вывод как команда ls *-dgils.*

-print — показывает на экране найденные файлы.

-exec command {} ; — выполняет над найденным файлом указанную команду; обратите внимание на синтаксис.

-ok — перед выполнением команды указанной в *-exec*, выдаёт запрос.

-depth — начинать поиск с самых глубоких уровней вложенности, а не с корня каталога.

-prune — используется, когда вы хотите исключить из поиска определённые каталоги.

N — количество дней.

Глава 5

Базы данных

5.1 PostgreSQL

5.1.1 Установка pgAdmin4 в Deepin linux

Кроссплатформенное приложение **pgAdmin 4** является одним из клиентов, предназначенных для серверов **PostgreSQL**. С помощью **pgAdmin** очень удобно администрировать сервера **PostgreSQL**, а так же создавать базы данных. **pgAdmin 4** основан на Python и по сути запускает на компьютере свой веб серер на порту 5050.

Однако для установки на Debian/Ubuntu нет готовых deb пакетов и это требует некоторых усилий для его установки.

Установка pgAdmin 4

1. Сначала необходимо обновить кэш и установить необходимые для запуска **pgAdmin 4** пакеты:

```
$ sudo apt update
$ sudo apt install build-essential libssl-dev libffi-dev libgmp3-dev \
                   virtualenv python-pip libpq-dev python-dev
```

2. Теперь необходимо создать виртуальное окружение

```
$ cd ~
$ virtualenv pgadmin4
$ cd pgadmin4
```

3. Теперь необходимо активировать виртуальное окружение

```
$ source bin/activate
```

4. Подготовительные операции произведены, теперь нужно скачать последнюю версию приложения. На момент написания статьи - это версия `pgadmin4-3.6`. Чтобы скачать, переходим по ссылке.

5. Копируем в буфер обмена ссылку на файл `pgadmin4-2.1-py2.py3-none-any.whl` и загружаем при помощи `wget` или `curl` в созданную директорию `pgadmin4`.

```
$ wget https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v3.6/pip/pgadmin4-3.6-py2.py3-none-any.whl
```

6. И теперь, собственно, установка **pgAdmin 4**

```
$ pip install pgadmin4-3.6-py2.py3-none-any.whl
```

7. После установки необходимо создать конфигурационный файл:

```
$ nano lib/python2.7/site-packages/pgadmin4/config_local.py
```

И записать в него следующее содержимое:

```
import os
DATA_DIR = os.path.realpath(os.path.expanduser(u'~/.pgadmin/'))
LOG_FILE = os.path.join(DATA_DIR, 'pgadmin4.log')
SQLITE_PATH = os.path.join(DATA_DIR, 'pgadmin4.db')
SESSION_DB_PATH = os.path.join(DATA_DIR, 'sessions')
STORAGE_DIR = os.path.join(DATA_DIR, 'storage')
SERVER_MODE = False
```

8. Собственно после этого можно заустить **pgAdmin 4** командой:

```
$ python lib/python2.7/site-packages/pgadmin4/pgAdmin4.py
```

Примечание: В моём случае пострeбовалось доустановить ещё несколько модулей *python*. Полный список приведён ниже.

```
$ pip install flask-htmlmin flask-babelex flask-login flask-mail \
              flask-paranoid flask-security flask-gravatar flask-sqlalchemy \
              flask-migrate backports.csv sshtunnel sqlparse psycopg2-binary
```

Совет: На мой взгляд, запускать **pgAdmin 4** приведённой выше командой не учень удобно. Поэтому я создал в домашней директории папку `bin` и добавил в неё скрипт для запуска **pgAdmin**

```
$ mkdir ~/bin
$ echo python ~/pgadmin/lib/python2.7/site-packages/pgadmin4/pgAdmin4.py > ~/bin/pgadmin
```

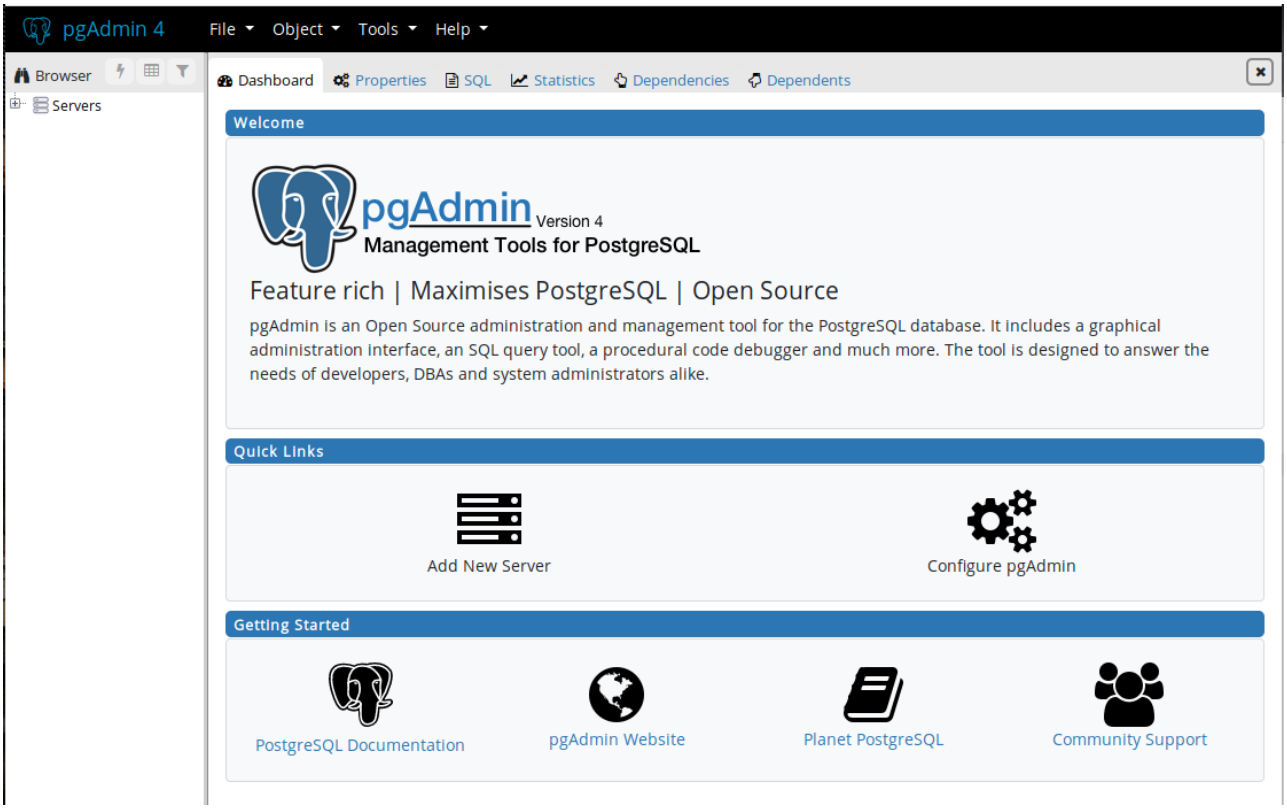
После создания файла достаточно перезайти в эмулятор терминала и можно пробовать запускать программу:

```
$ pgadmin
```

Если всё запускается без ошибок, то мы увидим что-то подобное:

```
PC:~$ pgadmin
Starting pgAdmin 4. Please navigate to http://127.0.0.1:5050 in your browser.
 * Serving Flask app "pgadmin" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
```

Как видно из сообщения, которое вывел скрипт, интерфейс **pgAdmin 4** доступен в броузере по адресу `http://127.0.0.1:5050`. Копируем ссылку, открываем броужен и переходим по этому адресу:



Удаление pgAdmin 4

Чтобы удалить pgAdmin, нужно выполнить следующие команды:

1. Перейти в директорию `pgadmin4`

```
$ cd ~/pgadmin4
```

2. Активировать виртуальное окружение

```
$ source bin/activate
```

3. Удалить **pgAdmin 4**

```
$ pip uninstall pgadmin4
```

После запуска комманда выдаст список удаляемых файлов и попросит подтвердить удаление:

```
Uninstalling pgadmin4-3.6:
  Would remove:
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4-3.6.dist-info/*
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/.editorconfig
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/.eslintignore
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/.eslintrc.js
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/.pycodestyle
    ...
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/webpack.test.config.js
    /home/mdima/pgadmin4/lib/python2.7/site-packages/pgadmin4/yarn.lock
Proceed (y/n)? y
  Successfully uninstalled pgadmin4-3.6
(pgadmin4)
```

4. Осталось только деактивировать виртуальное окружение.

```
deactivate
```

На этом всё.

5.1.2 Установка консольного клиента *pgcli* в Linux

pgcli - консоль для управления **Postgres**.

Умный инструмент командной строки для **Postgres** с автокомплитом и подсветкой синтаксиса написанный на **Python**.

Быстрая Установка

Если у вас уже установлен *Python*, то ,как правило, для установки должно хватить одной команды:

```
$ pip install pgcli
```

Возможно потребуется `sudo`, либо можно установить только для текущего пользователя с параметром `--user`.

```
$ pip install --user pgcli
```

Подробности установки при отсутствии утилиты *pip*

Если же окажется что пакетный менеджер для **Python**, под названием **pip** у вас ещё не установлен ии вы получили ошибку о том, что данная команда не найдена - не беда.

Для начала проверим, действительно ли **pip** не установлен в системе. Для этого надо ввести следующую команду:

```
$ which pip
```

В ответ мы должны получить абсолютный путь, что-то типа:

```
/usr/bin/pip
```

Если же вы получите ошибку, либо команда вообще ничего не выведет, значит **pip** отсутсвует в системе.

Установить его можно простой командой. Для систем на базе **Debian** (*Debian, Ubuntu, Mint и тд.*)

```
$ sudo apt-get install python-pip
```

Для систем на базе **Redhat** (*Fedora, Centos, RHEL и тд.*)

```
$ sudo yum install python-pip
```

pgcli использует *psycopg* для подключения к базе данных *postgres*. Для установки *psycopg*, вам потребуется чтобы в системе были установлены *libpq* и *python-dev*. Поскольку *psycopg* это *C* расширение для *Python*, необходимо чтобы так же был установлен компилятор *C*.

```
$ sudo apt-get install libpq-dev python-dev    # debian
$ sudo yum install postgresql-devel python-devel  # redhat
```

Теперь, когда установлены все зависимости, можно приступить к установке **pgcli**.

```
$ sudo pip install pgcli
```

Примечание: Возможно при запуске **pgcli** будет ругаться, что пакет *psycopg2* будет переименован в релизе 2.8

Будет выдаваться ссобщение типа:

```
$ pgcli
/home/user/.local/lib/python2.7/site-packages/psycopg2/__init__.py:144:
 UserWarning: The psycopg2 wheel package will be renamed from release 2.8;
 in order to keep installing from binary please use "pip install psycopg2-binary" instead.
 For details see:
 <http://initd.org/psycopg/docs/install.html#binary-install-from-pypi>.
```

В этом случае необходимо установить бинарники *psycopg2*, как это рекомендуется в сообщении:

```
$ pip install psycopg2-binary
```

На этом всё.

Глава 6

DevOps

6.1 Ansible

6.1.1 Использование Vault – зашифрованного хранилища

Начиная с версии 1.5 в **Ansible** была доабвлена возможность хранения секретных данных, таких как пароли или RSA ключи, в зашифрованных файлах – *vault* ("хранилище").

Для работы с такими хранилищами используется утилита ansible-vault, которая принимает два основных аргумента – `--ask-vault-pass` или `--vault-password-file`, которые могут быть заданы через `ansible.cfg`.

Располагается в `/usr/bin/ansible-vault` и представлет собой *Python*-скрипт.

Шифрование файлов

Для создания хранилища – используется `create`:

```
$ ansible-vault create file.yml
New Vault password:
Confirm New Vault password:
```

После чего открывается редактор по умолчанию, что бы в файл можно было добавить данные.

Результат:

```
$ cat file.yml
$ ANSIBLE_VAULT;1.1;AES256
32333262313562656230663934383566373966633138346137313937306230373633333730343839
63303461666262353765303336626639616363653936313340a623165306231383636643066306665
39303633623361643065636338326336333366656363313762316336623732626631643437336663
64663362323161313390a3164643838376233613339643832356163653162633365653361366630636
6635
```

Что бы зашифровать уже имеющий файл – используется `encrypt`:

```
$ echo "data" > raw_file.txt
$ ansible-vault encrypt raw_file.txt
New Vault password:
Confirm New Vault password:
Encryption successful
$ cat raw_file.txt
$ ANSIBLE_VAULT;1.1;AES256
65303737323263313938656134323635313039363131626331643835393034323335343432383438
32326431623135346432316136616232646563656634376 50a3536613762353461343133623653066
64393066333937373334353361623735366334666463346633613463373532646535636566663831
34346232356539336 50a313038343833323236623061633065323266336237623461383930646662
6538
```

Расшифровка файлов

Что бы зашифрованный файл снова вернуть в plaintext – используем `decrypt`:

```
$ ansible-vault decrypt file.yml
Vault password:
Decryption successful
$ cat file.yml
test data
```

Редактирование файлов

Используется `ansible-vault edit`:

```
$ ansible-vault edit file.yml
Vault password:
```

Файл расшифровывается во временный файл, открывается для редактирования, после завершения – снова шифруется и сохраняется в старом месте.

Просмотр файла

Что бы просто просмотреть содержимое хранилища – используем `view`.

Шифруем файл опять:

```
$ ansible-vault encrypt file.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

И проверяем содержимое:

```
$ ansible-vault view file.yml
Vault password:
test data
```

Смена пароля

Для смены пароля используется `rekey`, который можно применять к нескольким файлам сразу, если они зашифрованы одним ключём (паролем).

Создадим новый:

```
$ ansible-vault create file2.yml
New Vault password:
Confirm New Vault password:
```

И меняем пароль для обоих файлов:

```
$ ansible-vault rekey file.yml file2.yml
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

Примеры

Пример использования зашифрованного файла

Создаём простой плейбук `helloworld.yml`:

```
---
- hosts: all
tasks:
  - debug:
      msg: "hello world"
```

Проверяем:

```
$ ansible-playbook -i "localhost," -c local helloworld.yml
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [localhost]
TASK [debug] ****
ok: [localhost] => {
"msg": "hello world"
}
PLAY RECAP ****
localhost                  : ok=2    changed=0    unreachable=0    failed=0
```

OK, теперь добавим создание файла с текстом из зашифрованного файла в Ansible.

Создаём файл `vars/exmple_text.yml` со строкой:

```
data_text: |
  some text data
  another text data
```

Создаём файл с паролем:

```
$ echo ‘‘MySuperPass’’ > ~/.ansible_pass.txt
```

Шифруем файл `vars/exmple_text.yml`:

```
$ ansible-vault encrypt vars/exmple_text.yml --vault-password-file ~/.ansible_pass.txt
Encryption successful
```

Проверяем его:

```
$ head -n 2 vars/exmple_text.yml
$ ANSIBLE_VAULT;1.1;AES256
33303034383034313365383135323837613963646235393533643361343061306336373236663837
```

Обновляем наш `helloworld.yml`, добавляем создание файла, который будет содержать текст :

```
---
- hosts: all
  tasks:
    - debug:
        msg: "hello world"
    - name: add var
      include_vars:
        file: exmple_text.yml
    - name: Add descrypted text from exmple_text.yml
      copy:
        content="{{ data_text }}"
        dest=/tmp/example_text_out.txt
```

Запускаем, передавая `--vault-password-file`:

```
$ ansible-playbook -i "localhost," -c local helloworld.yml --vault-password-file ~/.ansible_pass.
↪txt
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [localhost]
TASK [debug] ****
ok: [localhost] => {
"msg": "hello world"
}
TASK [add var] ****
ok: [localhost]
TASK [Add descrypted text from exmple_text.yml] ****
changed: [localhost]
PLAY RECAP ****
localhost                  : ok=4    changed=1    unreachable=0    failed=0
```

Проверяем:

```
$ cat /tmp/example_text_out.txt
some text data
another text data
```

Всё на месте.

Пример шифрования переменной

Кроме файла – можно зашифровать строку, например:

```
$ ansible-vault encrypt_string
New Vault password:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Confirm New Vault password:
Reading plaintext input from stdin. (ctrl-d to end input)
This is Text String
!vault |
$ ANSIBLE_VAULT;1.1;AES256
383639653435633539626662646463376134646632636636326262643735636334303236333566639
373733323339366233653337666133316365303533333437 0a373565343330633537363563656430
373636326366643538643535326330303262313562386436346230333965396561646664373435 65
396465303334313662 0a356438316635663561313665323739353766383233656261646538616165
663132376336356462656536333236353338616365393139373433636666539366465
Encryption successful
```

Либо выполнить одной командой:

```
$ ansible-vault encrypt_string "This is Text String" --name "encrypted_data_string" --vault-
↪password-file ~/.ansible_pass.txt
encrypted_data_string: !vault |
$ ANSIBLE_VAULT;1.1;AES256
366531626564343733623964643530613433373438663234363661386362666661656361636233 37
333934383338306430303634666161623539656335636263 0a363430313566323537363964663636
323039313335616336323232653461363231396165363434666637366661383336383061336539 35
306664373065383130 0a663735333639343064666636336139336536353734343963313032373338
356436396431366231633466626237636337633664623964333231333065326630 38
Encryption successful
```

Далее эта строка может использоваться в переменной.

Добавим новый файл `vars/strings.yml` с переменной `encrypted_data_string`:

```
---
encrypted_data_string: !vault |
            $ANSIBLE_VAULT;1.1;AES256
            366531626564343733623964643530613433373438663234363661386362666661656361636233 37
            333934383338306430303634666161623539656335636263 0a363430313566323537363964663636
            323039313335616336323232653461363231396165363434666637366661383336383061336539 35
            306664373065383130 0a663735333639343064666636336139336536353734343963313032373338
            356436396431366231633466626237636337633664623964333231333065326630 38
```

Обновим плейбук:

```
---
- hosts: all
    tasks:
      - debug:
          msg: "hello world"
      - name: add vars
        include_vars:
          file: exmple_text.yml
      - name: add another vars
        include_vars:
          file: strings.yml
      - name: Add decrypted text from exmple_text.yml
        copy:
          content="{{ data_text }}"
          dest=/tmp/example_text_out.txt
      - name: Add another decrypted text from strings.yml
        copy:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        content="{{ encrypted_data_string }}"
        dest=/tmp/another_example_text_out.txt
```

Запускаем выполнение:

```
$ ansible-playbook -i "localhost," -c local helloworld.yml --vault-password-file ~/.ansible_pass.
↪txt
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [localhost]
TASK [debug] ****
ok: [localhost] => {
"msg": "hello world"
}
TASK [add vars] ****
ok: [localhost]
TASK [add another vars] ****
ok: [localhost]
TASK [Add decrypted text from exmple_text.yml] ****
ok: [localhost]
TASK [Add another decrypted text from strings.yml] ****
changed: [localhost]
PLAY RECAP ****
localhost                  : ok=6    changed=1    unreachable=0    failed=0
```

Проверяем:

```
$ cat /tmp/another_example_text_out.txt
This is Text String
```

Или используя `--ask-vault-pass`:

```
$ ansible-playbook -i ‘‘localhost,’’ -c local helloworld.yml --ask-vault-pass
Vault password:
...
```

vault-id

В Ansible 2.4 и выше вместо `--ask-vault-pass` и `--vault-password-file` можно использовать `--vault-id`, который позволяет использовать разные пароли для разных файлов.

Создадим новые файлы вместо `exmple_text.yml` и `strings.yml`, зашифруем их разными паролями.

Файлы паролей:

```
$ echo "pass1" > pass1.txt
$ echo "pass2" > pass2.txt
```

Создаём файл `vars/data_text1.yml` с переменной *data_ text1*:

```
$ echo ‘‘data_text1: Data Text One’’ > vars/data_text1.yml
```

Шифруем его паролем из файла `pass1.txt`:

```
$ ansible-vault --vault-id pass1.txt encrypt vars/data_text1.yml
Encryption successful
```

Аналогично – второй файл:

```
$ echo ‘‘data_text2: Data Text Two’’ > vars/data_text2.yml
```

И тоже шифруем его, вторым паролем:

```
$ ansible-vault --vault-id pass2.txt encrypt vars/data_text2.yml
Encryption successful
```

Обновим плейбук:

```
---
- hosts: all
  tasks:
    - debug:
        msg: "hello world"
    - name: add data_text1.yml
      include_vars:
        file: data_text1.yml
    - name: add data_text2.yml
      include_vars:
        file: data_text2.yml
    - name: Add decrypted data_text1
      copy:
        content="{{ data_text1 }}"
        dest=/tmp/data_text1_out.txt
    - name: Add decrypted data_text2
      copy:
        content="{{ data_text2 }}"
        dest=/tmp/data_text2_out.txt
```

И запускаем его:

```
$ ansible-playbook -i "localhost," -c local --vault-id pass1.txt --vault-id pass2.txt helloworld.
↪yml
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [localhost]
TASK [debug] ****
ok: [localhost] => {
"msg": "hello world"
}
TASK [add data_text1.yml] ****
ok: [localhost]
TASK [add data_text2.yml] ****
ok: [localhost]
TASK [Add decrypted data_text1] ****
changed: [localhost]
TASK [Add decrypted data_text2] ****
changed: [localhost]
PLAY RECAP ****
localhost                  : ok=6    changed=2    unreachable=0    failed=0
```

Проверяем:

```
$ cat /tmp/data_text*
Data Text OneData Text Two
```

Так же `--vault-id` можно использовать вместе с `--vault-password-file`:

```
$ rm /tmp/data_text*
$ ansible-playbook -i "localhost," -c local --vault-id pass1.txt --vault-password-file pass2.txt␣
↪helloworld.yml
...
localhost                  : ok=6    changed=2    unreachable=0    failed=0
```

Либо использовать `--ask-vault-pass` вместо указания файла:

```
$ rm /tmp/data_text*
$ ansible-playbook -i "localhost," -c local --vault-id pass1.txt --ask-vault-pass helloworld.yml
Vault password:
...
localhost                  : ok=6    changed=2    unreachable=0    failed=0
```

Сам `--vault-password-file` тоже можно использовать несколько раз:

```
$ ansible-playbook -i "localhost," -c local --vault-password-file pass1.txt --vault-password-file␣
↪pass2.txt helloworld.yml
```

Ansible попробует каждый из переданных паролей, пока один из них не сработает для зашированного файла.

6.2 Язык программирования *Go*

6.2.1 Установка Go в debian

Установка.

1. Скачать архив с последней версией для Linux по ссылке.
2. Распаковать архив с в `/usr/local` созданием директории `/usr/local/go`. Пример:

```
$ sudo tar -C /usr/local -xzf go1.11.5.linux-amd64.tar.gz
```

3. Добавить путь `/usr/local/go/bin` файл профиля `~/.profile` или в файл `/etc/profile`. Необходимо добавить следующую строку:

```
1 export PATH=$PATH:/usr/local/go/bin
```

После этого необходимо перезайти в профиль, либо обновить переменные окружение командой:

```
$ source ~/.profile
```

Проверка правильности установки.

Создадим простое приложение *Hello World*. Создайте рабочую директорию для исходников, к примеру `$HOME/go` и в ней директории `src/hello`:

```
$ mkdir $HOME/go/src/hello
```

Перейдите в эту директорию и создайте файл `hello.go`:

```
$ cd $HOME/go/src/hello
$ touch hello.go
```

Откройте этот файл в любом текстовом редакторе и добавьте следующий код:

```
package main

import "fmt"

func main() {
   fmt.Printf("hello, world\n")
}
```

Следующая команда соберёт исполняемый файл `hello` в директории с созданным файлом:

```
$ go build hello.go
```

Проверим создался ли исполняемый файл:

```
$ ls -lh
итого 1,9M
-rwxr-xr-x 1 mdima mdima 1,9M фев 17 20:23 hello
-rw-r--r-- 1 mdima mdima   74 фев 17 20:21 hello.go
```

Запустим приложение. Если всё было установлено корректно и приложение собралось, то увидим следующее:

```
$ ./hello
hello, world
```

6.3 terraform

6.3.1 Установка *terraform* в Debian

https://www.terraform.io/

Установка *terraform* очень проста. Поскольку он распространяется в виде одного единственного исполняемого файла.

Для начала необходимо скачать по ссылке архив с последней версией *terraform* под свою операционную систему. В данном случае это будет *Linux 64-bit*.

```
$ wget https://releases.hashicorp.com/terraform/0.11.11/terraform_0.11.11_linux_amd64.zip
```

Распаковываем исполняемый файл в директорию `/usr/bin/`, либо в директорию `~/bin/` текущего пользователя, если таковая существует.

```
$ sudo unzip terraform_0.11.11_linux_amd64.zip -d /usr/bin/
```

По сути, процесс установки на этом закончен. Теперь можно проверить установленную версию:

```
$ terraform -v
Terraform v0.11.11
```

На этом всё.

6.3.2 Установка провайдера *VirtualBox* для *terraform*

Перед установкой провайдера *VirtualBox* необходимо установить компилятор языка *Go*, как описано по ссылке.

Установка производится следующей командой:

```
$ go get github.com/terra-farm/terraform-provider-virtualbox
```

Пример конфигурации:

```
resource "virtualbox_vm" "node" {
    count = 2
    name = "${format("node-%02d", count.index+1)}"

    image = "~/ubuntu-15.04.tar.xz"
    cpus = 2
    memory = "512mib"

    network_adapter {
        type = "nat"
    }

    network_adapter {
        type = "bridged"
        host_interface = "en0"
    }

    optical_disks = ["./cloudinit.iso"]
}

output "IPAddr" {
    # Get the IPv4 address of the bridged adapter (the 2nd one) on 'node-02'
    value = "${element(virtualbox_vm.node.*.network_adapter.1.ipv4_address, 1)}"
}
```

Примечание: Полезные ссылки: https://github.com/terra-farm/terraform-provider-virtualbox

6.4 Vagrant

6.4.1 Установка Vagrant на *Debian 9*

Vagrant является инструментом для создания и распространения среды разработки. **Vagrant** может работать с *VirtualBox/VMware*, а так же *AWS/OpenStack*, или в контейнерах (например *Docker/LXC*).

Сайт проекта: https://www.vagrantup.com/

Для начала работы с **Vagrant** необходимо установить VirtualBox.

6.4.2 *Vagrant*: Основные команды.

Посмотреть версию установленного *vagrant*:

```
$ vagrant --version
Vagrant 2.2.3
```

или:

```
$ vagrant -v
Vagrant 2.2.3
```

Так же можно посмотреть установленную и доступную для обновления версию:

```
$ vagrant version
Installed Version: 2.2.3
Latest Version: 2.2.3

You're running an up-to-date version of Vagrant!
```

Инициализация *vagrant* в текущей директории:

```
$ vagrant init
A `Vagrantfile` has been placed in this directory. You are now ready to
`vagrant up` your first virtual environment! Please read the comments in
the Vagrantfile as well as documentation on `vagrantup.com` for more
information on using Vagrant.
```

Эта команда создаёт в текущей директории конфигурационный файл `Vagrantfile` с дэфолтными настройками.

Перед первым запуском необходимо открыть этот файл и заменить значение переменной в строке `config.vm.box = "base"` с *base* на версию образа, который вы желаете запустить. К примеру:

```
1 config.vm.box = "debian/stretch64"
```

После сохранения запускаем бокс:

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'debian/stretch64'...
Progress: 90%
...
VirtualBox Guest Additions: Starting.
Unmounting Virtualbox Guest Additions ISO from: /mnt
==> default: Checking for guest additions in VM...
==> default: Installing rsync to the VM...
==> default: Rsyncing folder: /mnt/data/vagrant/test/ => /vagrant

==> default: Machine 'default' has a post `vagrant up` message. This is a message
==> default: from the creator of the Vagrantfile, and not from Vagrant itself:
==> default:
==> default: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports
```

Эта программ скачивает темплейт образа, если он ещё не был скачан вами ранее, создаёт виртуальный бокс и запускает его.

Информация об уже загруженных темплейтах можно посмотреть так:

```
$ vagrant box list
centos/7         (virtualbox, 1811.02)
debian/stretch64 (virtualbox, 9.6.0)
ubuntu/bionic64  (virtualbox, 20181219.0.0)
```

vagrant up vagrant ssh vagrant destroy vagrant halt vagrant suspend vagrant resume vagrant reload

vagrant status

```
$ vagrant halt
==> default: Attempting graceful shutdown of VM...
```

Установка гостевых дополнений

```
$ vagrant vbguest
[default] GuestAdditions 6.0.2 running --- OK.
```

Получить состояние бокса

```
$ vagrant status
Current machine states:

default                   running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to shut it
down forcefully, or you can run `vagrant suspend` to simply suspend the
virtual machine. In either case, to restart it again, simply run `vagrant
up`.
```

Получение информации обо всех боксах созданных *Vagrant*.

```
$ vagrant global-status
id       name    provider   state    directory
------------------------------------------------------------------------
0398ff9  omv5box virtualbox poweroff /mnt/mdima/vagrant/OpenMediaVault
b0b5f1f  default virtualbox running  /mnt/data/vagrant/debian_vb_test

The above shows information about all known Vagrant environments on this
machine. This data is cached and may not be completely up-to-date (use
"vagrant global-status --prune" to prune invalid entries). To interact with
any of the machines, you can go to that directory and run Vagrant, or you
can use the ID directly with Vagrant commands from any directory. For
example: "vagrant destroy 1a2b3c4d"
```

6.5 VirtualBox

6.5.1 Установка *VirtualBox* в *Debian 9 (Stretch)*.

Virtualbox - система виртуализации, на данный момент одна из наилучших бесплатных программных продуктов виртуализации. Но в *Debian 9 (Stretch)* он по-умолчанию отсутствует в стандартных

репозиториях.

Сайт проекта: https://www.virtualbox.org/

Установка из *Oracle Virtualbox third-party repository.*

Установку можно произвести из *Oracle Virtualbox third-party repository.*

Для начала необходимо подключить репозиторий.

Создадим файл `/etc/apt/source.list.d/virtualbox.list`:

```
$ sudo touch /etc/apt/source.list.d/virtualbox.list
```

Открываем файл в текстовом редакторе:

```
$ sudo nano /etc/apt/source.list.d/virtualbox.list
```

и добавляем строку:

```
1 deb http://download.virtualbox.org/virtualbox/debian stretch contrib
```

добавляем в систему *Oracle VirtualBox public key*

```
$ wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | sudo apt-key add -
$ wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- | sudo apt-key add -
```

обновляем кэш *apt*

```
$ sudo apt update
```

Если в процессе обновления не выдало никаких сообщений об ошибках или предупреждений, то можно проверить наличие доступных пакетов *virtualbox*

```
$ sudo apt-cache search virtualbox
virtualbox-5.0 - Oracle VM VirtualBox
virtualbox-5.1 - Oracle VM VirtualBox
virtualbox-5.2 - Oracle VM VirtualBox
virtualbox-6.0 - Oracle VM VirtualBox
```

Теперь можно установить любую из доступных версий *virtualbox*

```
$ sudo apt install virtualbox-6.0
```

Установка из deb-пакета

Можно так же просто скачать пакет *virtualbox* с сайта и установить его вручную.

Для начала заходим на страницу загрузки https://www.virtualbox.org/wiki/Linux_Downloads и копируем ссылку на нужную версию. В нашем случае для *Debian 9*.

Загружаем пакет:

```
$ wget https://download.virtualbox.org/virtualbox/6.0.2/virtualbox-6.0_6.0.2-128162~Debian~stretch_
↪amd64.deb
```

Устанавливаем пакет:

```
$ sudo dpkg -i virtualbox-6.0_6.0.2-128162~Debian~stretch_amd64.deb
```

На этом всё.