

# Оглавление

<b>1</b>	<b>Задачи</b>	<b>2</b>
1.1	Постановка задач . . . . .	2
1.2	Другие задачи . . . . .	2
<b>2</b>	<b>Работа с git</b>	<b>3</b>
2.1	Внесение коммитов на GitHub . . . . .	3
2.2	Использование репозитория GitHub для автоматизации процесса . . . . .	5
<b>3</b>	<b>Выполнение задач в консоли – на VM</b>	<b>6</b>
3.1	Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM . . . . .	6
3.1.1	создать LVM-том, добавить и смонтировать дополнительный диск для PostgreSQL в /var/lib/pgsql; . . . . .	6
3.1.2	Увеличить корневой раздел . . . . .	9
3.1.3	Поработать со snapshot-ами томов (создать, внести изменения, откатить); . . . . .	10
3.2	Задача №3 – Организация доступа . . . . .	14
3.3	Задача №4 – Работа с пакетным менеджером DNF . . . . .	15
<b>4</b>	<b>Выполнение задач – Использование скриптов</b>	<b>16</b>
4.1	Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM . . . . .	16
4.2	Задача №3 – Организация доступа . . . . .	19
4.3	Задача №4 – Работа с пакетным менеджером DNF . . . . .	21
4.4	Задача №5 – Инициализация и настройка PostgreSQL . . . . .	23
4.5	Задача №6 – Развёртывание Demo-базы и настройка доступа . . . . .	25
4.5.1	Дополнительно - Скрипт для создания новых пользователей в PostgreSQL и настройка прав доступа для них . . . . .	27
4.6	Задача №7 – Установка и работа в PgAdmin . . . . .	29
4.6.1	SQL-запрос, который считает количество вылетов из каждого аэропорта в заданном городе за июль. . . . .	29
4.6.2	SQL-запрос подсчитывает число вылетов из каждого аэропорта в указанном городе за июль и выводит вместе с ним сведения об аэропортах. . . . .	29
<b>5</b>	<b>Выполнение задач – Использование скриптов (дополнительные комментарии)</b>	<b>30</b>
5.1	Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM . . . . .	30
5.2	Задача №3 – Организация доступа (доп. комментарии) . . . . .	34
5.3	Задача №4 – Работа с пакетным менеджером DNF (доп. комментарии) . . . . .	39
<b>6</b>	<b>Дополнительно выполненные задачи</b>	<b>42</b>

# 1 Задачи

## 1.1 Постановка задач

1. Развернуть виртуальную машину с ОС Oracle Linux 8 любым удобным способом:
  - через портал «облака»;
  - вручную через консоль/CLI системы виртуализации;
  - автоматически с помощью Terraform/Ansible/Vagrant/API виртуализации;
2. Настройка дисковой подсистемы с помощью менеджера томов LVM:
  - создать LVM-тома, добавить и смонтировать дополнительный диск для PostgreSQL в `/var/lib/pgsql`;
  - увеличить корневой раздел;
  - поработать со snapshot-ами томов (создать, внести изменения, откатить);
3. Организовать доступ:
  - создание непривилегированных пользователей;
  - настройка `sudo` для выполнения привилегированных команд;
4. Работа с пакетным менеджером DNF:
  - Установка основных утилит из стандартных репозиториях Linux:
    - Редакторы и управление сессиями: `vim`, `nano`, `mc`, `screen`;
    - Сетевые клиенты и диагностика: `wget`, `curl`, `telnet`, `nmap-ncat`, `tcpdump`, `net-tools`, `bind-utils`;
    - Файловые системы и хранение: `autofs`, `nfs-utils`, `cloud-utils-growpart`, `lsuf`, `sysfsutils`, `sg3utils`;
    - Системный мониторинг и диагностика: `sysstat`;
    - Утилиты общего назначения: `pwgen`, `bc`, `unzip`, `glibc-langpack-ru`;
    - Разработка и контроль версий: `git`;
  - Установка модулей PostgreSQL версии 15;
5. Инициализация и настройка PostgreSQL:
  - создать базу данных PostgreSQL;
  - инициализировать кластер с включёнными контрольными суммами;
  - настроить удалённое подключение (правки `postgresql.conf`, `pg_hba.conf`);
6. Развёртывание Демо-базы и настройка доступа:
  - скачать Демо-базу, например, с:  
<https://postgrespro.ru/education/courses/DBA1>  
<https://postgrespro.ru/docs/postgrespro/16/demodb-bookings-installation>;
  - импортировать Демо-базу через SQL-дамп;
  - создать пользователя и выдать привилегии на созданные базы;
7. Установка и работа в PgAdmin:
  - установить PgAdmin Desktop;
  - настроить подключение к базе;
  - выполнить простые и сложные SQL-запросы (`WHERE`, `GROUP BY`, `ORDER BY`, `JOIN`);
  - проанализировать планы запросов через `EXPLAIN PLAN`;

## 1.2 Другие задачи

1. Проекты вести в GitLab/GitHub - код, инструкции и т.п
2. Автоматизировать как можно больше шагов/этап с помощью скриптов `bash`, Ansible

## 2 Работа с git

### 2.1 Внесение коммитов на GitHub

Использована утилита git на ОС archlinux. Версия программы: 2.50.0-1.

#### 0.1. Подготовка локальной папки с проектом

```
cd /путь/к/папке_проекта git init
```

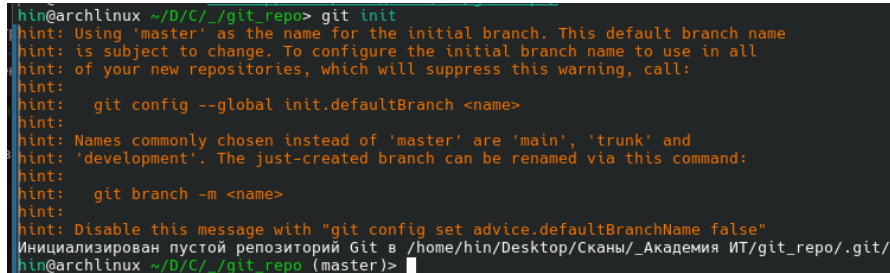
A terminal window showing the execution of the 'git init' command. The prompt is 'hin@archlinux ~/D/C/\_/git\_repo>'. The output shows several hints: 'Using 'master' as the name for the initial branch. This default branch name is subject to change. To configure the initial branch name to use in all of your new repositories, which will suppress this warning, call: git config --global init.defaultBranch <name>'. It also lists common branch names like 'main', 'trunk', and 'development'. Finally, it says 'Disable this message with "git config set advice.defaultBranchName false"'. The command completes successfully, showing '(master)>'.

Рис. 1: Создание локальной папки

#### 0.2 Задание для Git имени и почты

```
# Задаём имя
git config --global user.name "EfremovIlyaValentinovich"

# Задаём почту
git config --global user.email "burdgun@gmail.com"

# Посмотрим все ли верно (посмотрим настройки)
git config --list

# Дополнительно задам использовать утилиту "cat" как пейджер:
git config --global core.pager cat
```

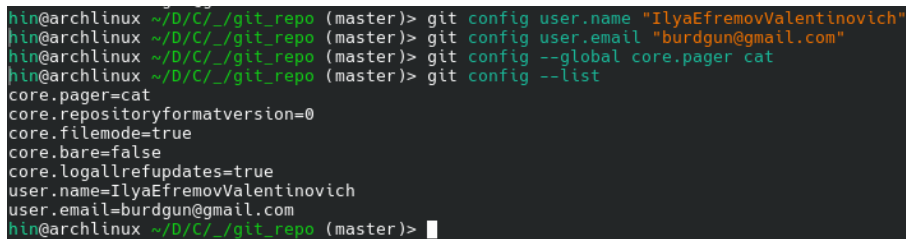
A terminal window showing a series of 'git config' commands and their output. The commands are: 'git config user.name "IlyaEfremovValentinovich"', 'git config user.email "burdgun@gmail.com"', 'git config --global core.pager cat', and 'git config --list'. The output of 'git config --list' shows the configured settings: 'core.pager=cat', 'core.repositoryformatversion=0', 'core.filemode=true', 'core.bare=false', 'core.logallrefupdates=true', 'user.name=IlyaEfremovValentinovich', and 'user.email=burdgun@gmail.com'. The prompt is 'hin@archlinux ~/D/C/\_/git\_repo (master)>'.

Рис. 2: Задание имени и почты для локального репозитория проекта

#### 0.3 Добавление файлов

```
# Будут добавлены все файлы
git add .

# Создаётся "снимок т.е. commit с комментарием"
git commit -m "my_comment"
```

```

hin@archlinux ~/D/C/_/git_repo (master)> git add .
hin@archlinux ~/D/C/_/git_repo (master)> git commit -m "01.07.2025 04:04 task2 completed"

[master (корневой коммит) b19dbf1] 01.07.2025 04:04 task2 completed
2 files changed, 138 insertions(+)
create mode 100644 "\320\220\320\262\321\202\320\276\320\274\320\260\321\202\320\270\320\267\320\260\321\206\320\270\321\217\02task/lvm_setup.sh"
create mode 100644 "\320\224\320\276\320\272\321\203\320\274\320\265\320\275\321\202\320\260\321\206\320\270\321\217.pdf"
hin@archlinux ~/D/C/_/git_repo (master)>

```

Рис. 3: Первый коммит

#### 0.4 Привяжем удаленный репозиторий т.е. свяжем с GitHub.com

```

# Привязка удаленного репозитория
git remote add origin https://github.com/Hinalril/AIT24_devops_BME.git

# Отправка коммита на GitHub (для имени ветки – master)
# опция -u необходима для установки связи локальной ветки с удаленной
# => в будущем достаточно использовать команду git push | git pull
git push -u origin master

# Далее было необходимо ввести:
# 1. Имя с сайта GitHub
# 2. Код токена (вместо пароля)

```

```

hin@archlinux ~/D/C/_/git_repo (master) [128]> git remote add origin https://github.com/Hinalril/AIT24_devops_BME.git

error: внешний репозиторий origin уже существует
hin@archlinux ~/D/C/_/git_repo (master) [3]> git push -u origin master
Username for 'https://github.com': Hinalril
Password for 'https://Hinalril@github.com':
Перечисление объектов: 6, готово.
Подсчет объектов: 100% (6/6), готово.
При сжатии изменений используется до 16 потоков
Сжатие объектов: 100% (4/4), готово.
Запись объектов: 100% (6/6), 498.34 КиБ | 49.83 МиБ/с, готово.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com:Hinalril/AIT24_devops_BME.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
hin@archlinux ~/D/C/_/git_repo (master)>

```

Рис. 4: Привязка и отправка данных на GitHub

#### 1.1. Дальнейшие коммиты

```

# Убедится, что изменения есть
git status

# Добавить изменения в индекс (все файлы)
git add .

# Либо можно выборочно выбрать файлы
git add путь/к/файлу_1 путь/к/файлу_2

# Далее совершить коммит и не забыть добавить к нему осмысленный комментарий
git commit -m "Understandable comment, not trash"

# Отправить коммит на GitHub
git push

```

#### 1.2. Проверка актуальности локального репозитория

```

# Синхронизируемся с удаленным репозиторием на случай изменений от других людей
git pull

```

#### 1.3. Просмотреть историю коммитов актуальности локального репозитория

```

git log --oneline --graph --decorate

```

## 2.2 Использование репозитория GitHub для автоматизации процесса

### 1.1. Клонирование удаленного репозитория к себе на компьютер

```
git clone https://github.com/Hinalril/AIT24_devops_BME.git
```

### 1.2. Переход в локальный репозиторий только что скачанный

```
cd AIT24_devops_BME/Auto
```

### 1.3. Добавим право исполнения для скриптов с помощью утилиты chmod

```
chmod +x 02_lvm_setup.sh  
chmod +x 03_user_setup.sh  
chmod +x 04_dnf_setup.sh
```

### 1.4. Запустим по очереди скрипты

```
sudo ./02_lvm_setup.sh  
sudo ./03_user_setup.sh  
sudo ./04_dnf_setup.sh
```

### 3 Выполнение задач в консоли – на VM

#### 3.1 Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM

##### 3.1.1 создать LVM-том, добавить и смонтировать дополнительный диск для PostgreSQL в /var/lib/postgresql;

###### Команды 1–5

```
# 1.1 инициализируем диск как Physical Volume
sudo pvcreate /dev/sda

# 1.2 добавляем его в существующую группу VG (например, ol_vbox)
sudo vgextend ol /dev/sda

# 1.3 создаём логический том 8 ГБ
sudo lvcreate -L 8G -n pglv ol

# 1.4 форматируем в XFS (или ext4)
sudo mkfs.xfs /dev/ol_vbox/pglv

# 1.5 монтируем и делаем постоянным
sudo mkdir -p /var/lib/postgresql
sudo echo '/dev/ol/pglv /var/lib/postgresql xfs defaults 0 2' >> /etc/fstab
sudo mount -a
```

```
hin@vbox:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda          8:0    0    8G  0 disk
sdb          8:16   0   20G  0 disk
├─sdb1       8:17   0    1M  0 part
├─sdb2       8:18   0    1G  0 part /boot
└─sdb3       8:19   0   19G  0 part
   └─ol_vbox-root 252:0   0   17G  0 lvm  /
      └─ol_vbox-swap 252:1   0    2G  0 lvm  [SWAP]
sr0         11:0    1 1024M  0 rom
```

Рис. 5: Посмотрим список доступных дисков (lsblk)

###### 1.1 Делаем новый диск PV

```
sudo pvcreate /dev/sda
```

```
hin@vbox:~$ sudo pvcreate /dev/sda
Physical volume "/dev/sda" successfully created.
```

Рис. 6: Результат pvcreate

###### 1.2 Включаем его в VG – Volume Group

```
sudo vgextend ol_vbox /dev/sda
```

```
hin@vbox:~$ sudo vgextend ol_vbox /dev/sda
Volume group "ol_vbox" successfully extended
```

Рис. 7: Расширенная VG ol\_vbox

### 1.3 Создаём логический том pglv на 5 ГБ

```
sudo lvcreate -L 5G -n pglv ol_vlox
```

```
hin@vbox:~$ sudo lvcreate -L 5G -n pglv ol_vlox
Logical volume "pglv" created.
hin@vbox:~$ lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINTS
sda	8:0	0	8G	0	disk	
└ol_vbox-pglv	252:2	0	5G	0	lvm	
sdb	8:16	0	20G	0	disk	
├sdb1	8:17	0	1M	0	part	
├sdb2	8:18	0	1G	0	part	/boot
├sdb3	8:19	0	19G	0	part	
└ol_vbox-root	252:0	0	17G	0	lvm	/
└ol_vbox-swap	252:1	0	2G	0	lvm	[SWAP]
sr0	11:0	1	1024M	0	rom	

Рис. 8: Проверяем изменения после lvcreate

### 1.4 Форматируем в XFS

```
# Сначала посмотрим, какой тип у диска (Рис. 5)
lsblk -o NAME,FSTYPE,SIZE
# Затем отформатируем логический диск (Рис. 6)
sudo mkfs.xfs /dev/ol_vbox/pglv
```

```
mkfs.xfs /dev/ol_vbox/pglv
```

- mkfs.xfs — утилита для создания файловой системы XFS на заданном блочном устройстве.
- /dev/ol\_vlox/pglv — LVM-том (логический том) с именем pglv, находящийся в группе томов ol\_vlox.

При выполнении этой команды:

- Утилита сотрёт все данные на томе /dev/ol\_vbox/pglv.
- Запишет метаданные XFS (супблок, журналы, таблицы аллокации и т.д.) в начало и в конец тома.
- Подготовит том так, чтобы ядро Linux могло смонтировать его как XFS-раздел.
- После окончания получен раздел, готовый к использованию под XFS.

```
hin@vbox:~$ lsblk -o NAME,FSTYPE,SIZE
```

NAME	FSTYPE	SIZE
sda	LVM2_member	8G
└ol_vbox-pglv		5G
sdb		20G
├sdb1		1M
├sdb2	xfs	1G
├sdb3	LVM2_member	19G
└ol_vbox-root	xfs	17G
└ol_vbox-swap	swap	2G
sr0		1024M

Рис. 9: Информация о диске перед форматированием

```

hin@vbox:~$ sudo mkfs.xfs /dev/ol_vbox/pglv
[sudo] пароль для hin:
meta-data=/dev/ol_vbox/pglv      isize=512    agcount=4, agsize=327680 blks
       =                       sectsz=512    attr=2, projid32bit=1
       =                       crc=1        finobt=1, sparse=1, rmapbt=1
       =                       reflink=1    bigtime=1 inobtcount=1 nrext64=1
data      =                       bsize=4096   blocks=1310720, imaxpct=25
       =                       sunit=0      swidth=0 blks
naming    =version 2              bsize=4096   ascii-ci=0, ftype=1, parent=0
log       =internal log          bsize=4096   blocks=16384, version=2
       =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime  =none                  extsz=4096   blocks=0, rtextents=0

```

Рис. 10: Форматирование тома в XFS (mkfs.xfs)

### 1.5 Монтируем и делаем постоянным

```

# создаём точку монтирования
sudo mkdir -p /var/lib/pgsql

# добавляем строку в /etc/fstab для автомонтирования
echo '/dev/ol/pglv /var/lib/pgsql xfs defaults 0 2' | sudo tee -a /etc/fstab

# перезагружаем таблицу юнитов
systemctl daemon-reload

# монтируем всё из fstab
sudo mount -a

```

```

hin@vbox:/dev$ echo 'dev/ol_vlox/pglv /var/lib/pgsql xfs defaults 0 2' | sudo tee -a /etc/fstab
dev/ol_vlox/pglv /var/lib/pgsql xfs defaults 0 2

```

Рис. 11: Успешное добавление записи и монтирование тома

```

hin@vbox:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda                                  8:0      0   8G  0 disk
└─ol_vbox-pglv                      252:2    0    5G  0 lvm  /var/lib/pgsql
sdb                                  8:16     0  20G  0 disk
├─sdb1                              8:17     0   1M  0 part
├─sdb2                              8:18     0    1G  0 part /boot
├─sdb3                              8:19     0   19G  0 part
└─ol_vbox-root                      252:0    0   17G  0 lvm  /
    └─ol_vbox-swap                  252:1    0    2G  0 lvm  [SWAP]
sr0                                  11:0     1 1024M  0 rom

```

Рис. 12: Результат: Создание LVM-тома под /var/lib/pgsql



### 3.1.2 Увеличить корневой раздел

#### 2.1. Посмотрим на ситуацию до увеличения

```
# Посмотрим текущий размер LV
lsblk -f | grep root

# Посмотрим использование диска для корневой точки /
df -h /
```

```
hin@vbox:~$ lsblk -f | grep root
└─ol_vbox-root xfs                    528b8452-f5e5-48a1-a3f9-14f225ec6f47    12,5G    26% /
hin@vbox:~$ sudo df -h /
Файловая система    Размер  Использовано  Дст  Использовано%  Смонтировано в
/dev/mapper/ol_vbox-root  17G      4,5G        13G      27% /
```

Рис. 13: Статус до расширения

#### 2.2. Посмотрим сколько есть свободного места

```
sudo vgs
# команда vgs выводит сводную таблицу по всему группам томов
# VFree – доступно, т.е. не распределено ни одному LV.
```

```
hin@vbox:~$ sudo vgs
[sudo] пароль для hin:
VG      #PV #LV #SN Attr   VSize  VFree
ol_vbox  2   3   0 wz--n- 26,99g <3,00g
```

Рис. 14: Видим, что доступно меньше 3g

#### 2.3. Увеличим логический том root LV

```
sudo lvextend -L +2G -r /dev/ol_vbox/root
```

- lvextend – расширяет существующий LV
- -L +2G – добавляет +2 ГБ к текущему размеру тома
- -r – автоматически увеличивает файловую систему, находящуюся внутри LV
- /dev/ol\_vbox/root – путь к корневому логическому тому

```
hin@vbox:~$ sudo lvextend -L +2G -r /dev/ol_vbox/root
[sudo] пароль для hin:
File system xfs found on ol_vbox/root mounted at /.
Size of logical volume ol_vbox/root changed from <17,00 GiB (4351 extents) to <19,00 GiB (4863 extents).
Extending file system xfs to <19,00 GiB (20396900352 bytes) on ol_vbox/root...
xfs_growfs /dev/ol_vbox/root
meta-data=/dev/mapper/ol_vbox-root isize=512    agcount=4, agsize=1113856 blks
=                               sectsz=512    attr=2, projid32bit=1
=                               crc=1         finobt=1, sparse=1, rmapbt=1
=                               reflink=1      bigtime=1 inobtcount=1 nrext64=1
=                               exchange=0
data      =                       bsize=4096    blocks=4455424, imaxpct=25
=                       sunit=0             swidth=0 blks
naming    =version 2              bsize=4096    ascii-ci=0, ftype=1, parent=0
log       =internal log          bsize=4096    blocks=16384, version=2
=                       sectsz=512          sunit=0 blks, lazy-count=1
realtime  =none                  extsz=4096    blocks=0, rtextents=0
data blocks changed from 4455424 to 4979712
xfs_growfs done
Extended file system xfs on ol_vbox/root.
Logical volume ol_vbox/root successfully resized.
```

Рис. 15: Результат увеличения логического тома root

## 2.4. Посмотрим на ситуацию после увеличения

```
# Посмотрим текущий размер LV
lsblk -f | grep root

# Посмотрим использование диска для корневой точки /
df -h /
```

```
hin@vbox:~$ lsblk -f | grep root
└─ol_vbox-root xfs                    528b8452-f5e5-48a1-a3f9-14f225ec6f47    14,4G    24% /
└─ol_vbox-root xfs                    528b8452-f5e5-48a1-a3f9-14f225ec6f47    14,4G    24% /
hin@vbox:~$ df -h /
Файловая система      Размер  Использовано  Дрст  Использовано%  Смонтировано в
/dev/mapper/ol_vbox-root 19G      4,6G          15G      24% /
```

Рис. 16: Результат увеличения логического тома root

## 3.1.3 Поработать со snapshot-ами томов (создать, внести изменения, откатить);

## Опр. Snapshot

Снапшот (snapshot) — это «моментальный снимок» состояния логического тома в момент его создания. LVM создаёт «копию-только-для-чтения» оригинального тома, но при этом не дублирует все данные, а хранит только изменения («копия-при-записи» или CoW).

В момент создания снапшота резервируется (например) 1 ГиБ CoW-области. Снапшот и исходный LV делят одни и те же блоки — место почти не расходуется.

В момент записи в какой-то блок исходного LV, происходят изменения. Старая версия изменяемого блока копируется в CoW область снапшота, а новая версия пишется в исходный LV.

Расход места. Допустим, мы изменяем 4MiB памяти на LV (как самая минимальная ячейка для XFS или Ext4). Тогда мы тратим 4MiB из CoW памяти.

Пока сумма изменений будет меньше объема CoW ( $\leq 1$  ГиБ) — снапшот будет работать корректно. Если изменений будет больше, CoW область переполнится: снапшот перейдет в состояние Invalid (т.е. потерянные данные).

Снапшот желательно хранить в той же VG, что и основной том, чтобы LVM мог эффективно управлять пространством.

## Устройство LVM-пирога

Диски/разделы → PV → VG → LV → Файловая система → каталог / приложение

Уровень	Что это	Главная роль
<b>PV</b> (Physical Volume)	Физический носитель: диск <code>/dev/sdb</code> , раздел <code>/dev/sda3</code> , loop-файл и т.д.	«Кирпичи» пула; содержит физические экстенды LVM.
<b>VG</b> (Volume Group)	Объединение одного или нескольких PV в общий «пул» (пример: <code>ol_vbox</code> ).	Даёт единый резерв пространства; можно расширять, добавляя новые PV.
<b>LV</b> (Logical Volume)	Логический том, «вырезанный» из VG ( <code>root</code> , <code>home</code> , <code>pglv</code> ).	Ведёт себя как раздел: форматируется, монтируется; легко расширяется, снапшотится.

Таблица 1: Структура LVM: PV → VG → LV

## Желаемое место для Snapshot'a

Размер snapshot должен перекрывать объем ожидаемых изменений за время жизни снапшота, т.к. если память CoW области переполнится, снимок помечается как invalid и откат становится невозможным.

## Устройство LVM-пирога

Диски/разделы → PV → VG → LV → Файловая система → каталог / приложение

### 3.1. Проверяем свободное место в группе томов (VG) ol\_vbox

```
sudo vgs
```

```
hin@vbox:~$ sudo vgs
[sudo] пароль для hin:
VG      #PV #LV #SN Attr   VSize  VFree
ol_vbox  2   3   0 wz--n- 26,99g 1020,00m
```

Рис. 17: Проверка свободного места до создания снапшота

### 3.2. Создаём снапшот корневого LV на 1 ГиБ

```
# Создание снапшота
sudo lvcreate -L 1G -s -n rootsnap /dev/ol_vbox/root

# убеждаемся, что rootsnap создан
sudo lvs -o lv_name,lv_size,data_percent,origin
```

```
hin@vbox:~$ sudo lvcreate -L 1G -s -n rootsnap /dev/ol_vbox/root
Volume group "ol_vbox" has insufficient free space (255 extents): 256 required.
hin@vbox:~$ sudo lvcreate -L 0.8G -s -n rootsnap /dev/ol_vbox/root
Rounding up size to full physical extent 820,00 MiB
Logical volume "rootsnap" created.
```

Рис. 18: Создание снапшота

### 3.3. Смотрим свободное место после создания снапшота

```
sudo vgs
```

```
hin@vbox:~$ sudo vgs
VG      #PV #LV #SN Attr   VSize  VFree
ol_vbox  2   4   1 wz--n- 26,99g 200,00m
```

Рис. 19: Проверяем изменение свободного места после создания снапшота

### 3.4. Монтируем снапшот «только-чтение»

```
# Создаём точку монтирования
# - опция -p выдаст ошибку, если каталог уже существует
sudo mkdir -p /mnt/rootsnap

# Монтирует LVM-снапшот rootsnap только для чтения (ro) по адресу /mnt/rootsnap.
# - путь /dev/ol_vbox/rootsnap — устройство-снапшот, созданное ранее.
sudo mount -o ro /dev/ol_vbox/rootsnap /mnt/rootsnap

Отмонтируем снапшот, когда просмотр/бэкап завершены.
sudo umount /mnt/rootsnap
```

```

hin@vbox:~$ sudo mkdir -p /mnt/rootsnap
hin@vbox:~$ sudo mount -o ro /dev/ol_vbox/rootsnap /mnt/rootsnap
mount: /mnt/rootsnap: wrong fs type, bad option, bad superblock on /dev/mapper/ol_vb
ox-rootsnap, missing codepage or helper program, or other error.
        dmesg(1) may have more information after failed mount system call.
hin@vbox:~$ sudo mount -o ro,nouuid /dev/ol_vbox/rootsnap /mnt/rootsnap
hin@vbox:~$ sudo umount /mnt/rootsnap
hin@vbox:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda                                  8:0      0   20G  0 disk
├─sda1                              8:1      0    1M  0 part
├─sda2                              8:2      0    1G  0 part /boot
├─sda3                              8:3      0   19G  0 part
│   └─ol_vbox-swap                  252:1    0    2G  0 lvm  [SWAP]
│       └─ol_vbox-root-real          252:3    0   19G  0 lvm
│           └─ol_vbox-root            252:0    0   19G  0 lvm /
│               └─ol_vbox-rootsnap    252:5    0   19G  0 lvm
sdb                                  8:16     0    8G  0 disk
├─ol_vbox-pglv                     252:2    0    5G  0 lvm  /var/lib/pgsql
├─ol_vbox-root-real                 252:3    0   19G  0 lvm
│   └─ol_vbox-root                   252:0    0   19G  0 lvm /
│       └─ol_vbox-rootsnap            252:5    0   19G  0 lvm
│           └─ol_vbox-rootsnap-cow    252:4    0  820M  0 lvm
│               └─ol_vbox-rootsnap    252:5    0   19G  0 lvm
sr0                                  11:0     1 1024M  0 rom

```

Рис. 20: Создание снапшота

### 3.5. Генерируем изменения (500 МБ «мусора» во /var/tmp)

```

sudo dd if=/dev/zero of=/var/tmp/bigfile bs=1M count=500 status=progress
sync

```

```

hin@vbox:~$ sudo dd if=/dev/zero of=/var/tmp/bigfile bs=1M count=500 status=progress
441450496 bytes (441 MB, 421 MiB) copied, 2 s, 206 MB/s
500+0 records in
500+0 records out
524288000 bytes (524 MB, 500 MiB) copied, 2.66966 s, 196 MB/s

```

Рис. 21: Генерация мусора на 500 мб

### 3.6. Смотрим сколько места занято в COW-области снапшота

```

# Поле Data% покажет заполняемость
sudo lvs -o lv_name,lv_size,data_percent,origin /dev/ol_vbox/rootsnap
sync

```

```

hin@vbox:~$ sudo lvs -o lv_name,lv_size,data_percent,origin /dev/ol_vbox/rootsnap
LV      LSize  Data%  Origin
rootsnap 820,00m 61,56  root

```

Рис. 22: Snapshot занят на 61.56%

## 3.7. Откат к моменту создания снимка

```
sudo lvconvert --merge /dev/ol_vbox/rootsnap
# после перезапуска root вернётся к «снимку» sudo reboot
```

```
hin@vbox:~$ sudo lvconvert --merge /dev/ol_vbox/rootsnap
Delaying merge since origin is open.
Merging of snapshot ol_vbox/rootsnap will occur on next activation of ol_vbox/root.
hin@vbox:~$ sudo vgs
VG      #PV #LV #SN Attr   VSize  VFree
ol_vbox  2   3   1 wz--n- 26,99g 200,00m
hin@vbox:~$ sudo lvs -o lv_name,lv_size,data_percent,origin /dev/ol_vbox/rootsnap
LV      LSize  Data%  Origin
[rootsnap] 820,00m 61,58  root
```

Рис. 23: Откат снимка. Как видим: память ещё не высвободилась

## 3.8. Проверяем что место вернулось и снимка нет

```
# VFree опять +1G
sudo vgs

# ничего не выведет — снимок удалён
sudo lvs | grep rootsnap
```

```
hin@vbox:~$ sudo vgs
[sudo] пароль для hin:
VG      #PV #LV #SN Attr   VSize  VFree
ol_vbox  2   3   0 wz--n- 26,99g 1020,00m
hin@vbox:~$ sudo lvs | grep rootsnap
```

Рис. 24: Результат после перезагрузки.

## 3.2 Задача №3 – Организация доступа

### Выполнение

```
# 1.1 разрешаем sudo для участников wheel

sudo visudo

Внутри находим строку:
# %wheel ALL=(ALL) ALL

Убираем знак комментария:
%wheel ALL=(ALL) ALL

# 1.2 создаём дополнительную группу guest (если ещё нет)

getent group guest || sudo groupadd guest

# 1.3 создаём пользователя alice и добавляем в wheel

sudo useradd -m -s /bin/bash alice
sudo passwd alice           # задаём пароль
sudo usermod -aG wheel alice # права sudo через wheel

# 1.4 создаём пользователя NyM и добавляем в guest

sudo useradd -m -s /bin/bash NyM
sudo passwd NyM           # задаём пароль
sudo usermod -aG guest NyM # нет sudo-прав, только guest

# 1.5 проверяем, кто состоит в группах wheel и guest

getent group wheel
getent group guest
```

Аналогично можно проделать для всех остальных пользователей, а не только для 1-го в каждой группе.

### 3.3 Задача №4 – Работа с пакетным менеджером DNF

#### Выполнение

```
# 1.1 — подключаем EPEL 8 и обновляем кеш репозиторияев
rpm -q oracle-epel-release-el8
sudo dnf -y install oracle-epel-release-el8
sudo dnf makecache -y

# 2.1 установка редакторов и управления сессиями
sudo dnf install -y vim-enhanced nano mc screen

# 2.2 установка сетевых клиентов и диагностики
sudo dnf install -y wget curl telnet nmap-ncat tcpdump net-tools bind-utils

# 2.3 установка файловых систем и средств хранения
sudo dnf install -y autofs nfs-utils cloud-utils-growpart lsof sysfsutils sg3utils

# 2.4 установка системного мониторинга
sudo dnf install -y sysstat

# 2.1 установка средств разработки и контроля версий
sudo dnf install -y git

# 3.1 подключение и установка модуля PostgreSQL 15
sudo dnf module reset -y postgresql
sudo dnf module enable -y postgresql:15
sudo dnf install -y postgresql-server postgresql

# 4.1 проверка установки пакетов
rpm -q vim-enhanced nano mc screen \
wget curl telnet nmap-ncat tcpdump net-tools bind-utils \
autofs nfs-utils cloud-utils-growpart lsof sysfsutils sg3utils \
sysstat pwgen bc unzip glibc-langpack-ru git \
postgresql-server postgresql
```

#### Примечание.

1. Репозиторий **OL 8** (Oracle Linux 8) не содержит RPM с именем **vim**, поэтому **vim** заменен на **vim-enhanced**;
2. Утилиты **screen** и **pwgen** отсутствуют в стандартных репозиториях. Для их установки был привлечен сторонний, неофициальный репозиторий: **EPEL 8**.

## 4 Выполнение задач – Использование скриптов

### 4.1 Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# lvm_setup.sh - автоматизация задания «LVM для PostgreSQL +-расширение root +-snapshot demo»
#
# Запустить от root:  sudo ./add_wheel_users.sh
#
# ПЕРЕД СТАРТОМ: при необходимости поправьте переменные в секции «НАСТРОЙКИ»

set -e

# -----
# НАСТРОЙКИ (редактируйте)
# -----
DISK="/dev/sdb"          # дополнительный диск под PostgreSQL
VG="ol"                  # существующая Volume Group (root расположена здесь)
PGLV_NAME="pglv"         # имя логического тома под /var/lib/pgsql
PGLV_SIZE="8G"           # размер тома под PostgreSQL
PG_MOUNT="/var/lib/pgsql"

ROOT_LV="root"           # имя корневого LV внутри $VG
ROOT_EXPAND="+1G"        # сколько добавить к корню

SNAP_SIZE="0.8G"         # объём CoW-области снапшота
TMP_FILE_MB=500          # сколько «мусора» писать для демонстрации (в МБ)
# -----

# -----
# Шаг 0. Требования запустить от root
# -----
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# ШАГ 1. PostgreSQL на отдельном LV
# -----
setup_pg_lv() {
    echo "=== ШАГ 1. Настройка тома под PostgreSQL ==="

    if ! pvs | grep -q "^${DISK}"; then
        pvcreate "${DISK}"
    else
        echo "Пропускаю pvcreate: диск ${DISK} уже инициализирован как PV."
    fi

    if ! vgdisplay "${VG}" | grep -q "${DISK}"; then
        vgextend "${VG}" "${DISK}"
    else
        echo "Пропускаю vgextend: диск уже входит в VG ${VG}."
    fi

    if ! lvs "${VG}/${PGLV_NAME}" &>/dev/null; then
        lvcreate -L "${PGLV_SIZE}" -n "${PGLV_NAME}" "${VG}"
    else
        echo "Пропускаю lvcreate: LV ${PGLV_NAME} уже существует."
    fi
}
```



```

fi

if ! blkid "/dev/${VG}/${PGLV_NAME}" &>/dev/null; then
    mkfs.xfs -f "/dev/${VG}/${PGLV_NAME}"
else
    echo "Пропускаю mkfs: файловая система уже есть на /dev/${VG}/${PGLV_NAME}."
fi

mkdir -p "${PG_MOUNT}"

fstab_entry="/dev/${VG}/${PGLV_NAME} ${PG_MOUNT} xfs defaults 0 2"
if ! grep -Fxq "${fstab_entry}" /etc/fstab; then
    echo "${fstab_entry}" >> /etc/fstab
else
    echo "Пропускаю запись в fstab: запись уже существует."
fi

mount -a
echo "Том под PostgreSQL примонтирован в ${PG_MOUNT}"
}

#
# ШАГ 2. Расширение корневого раздела
#
expand_root() {
    echo "=== ШАГ 2. Увеличение корневого LV на ${ROOT_EXPAND} ==="
    lvextend -r -L "${ROOT_EXPAND}" "/dev/${VG}/${ROOT_LV}"
    echo "Корневой LV расширен"
}

#
# ШАГ 3. Снапшот -> изменения -> откат
#
snapshot_demo() {
    echo "=== ШАГ 3. Демонстрация snapshot ==="

    if lvs "${VG}/rootsnap" &>/dev/null; then
        echo "Удаляю старый снапшот rootsnap"
        lvremove -f "/dev/${VG}/rootsnap"
    fi

    lvcreate -L "${SNAP_SIZE}" -s -n rootsnap "/dev/${VG}/${ROOT_LV}"
    mkdir -p /mnt/rootsnap
    mount -o ro,nouuid "/dev/${VG}/rootsnap" /mnt/rootsnap
    echo "Снапшот примонтирован в /mnt/rootsnap (только чтение)"

    echo "Создаю ${TMP_FILE_MB} МБ данных в /var/tmp для имитации изменений"
    dd if=/dev/zero of=/var/tmp/bigfile bs=1M count="${TMP_FILE_MB}" status=progress
    sync

    echo "Заполняемость CoW-области:"
    lvs -o lv_name,lv_size,data_percent,origin "/dev/${VG}/rootsnap"

    umount /mnt/rootsnap

    echo "Выполняю откат к снапшоту (lvconvert --merge)"
    lvconvert --merge "/dev/${VG}/rootsnap"
    echo
    echo "Внимание! Для завершения merge требуется перезагрузка."
    read -r -p "Перезагрузить систему сейчас? [y/N] " ans
    if [[ "${ans,,}" == "y" ]]; then
        reboot
    fi
}

```

```
else
    echo "Перезагрузку можно выполнить позже командой reboot"
fi
}

#-----
# ГЛАВНЫЙ БЛОК
#-----
setup_pg_lv
expand_root
snapshot_demo

echo "Сценарий завершён успешно"
```

## 4.2 Задача №3 – Организация доступа

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# user_setup.sh - автоматизация задания «LVM для PostgreSQL +-расширение root +-snapshot demo»
# sudo-правило для группы wheel
# создание группы guest
# создаёт пользователей, добавляя их в эту группу.
#
# Запустить от root: sudo ./add_wheel_users.sh

set -e

# -----
# Шаг -1. Требования запустить от root
# -----
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# Шаг 0. Подготовка
# -----
# 0.1. включаем правило для wheel в sudoers (с паролем)
if ! grep -Eq '^[:space:]*%wheel[:space:]+ALL' /etc/sudoers ; then
    sed -i 's/^[[:space:]]*%(wheel[:space:]).*ALL\)/\1/' /etc/sudoers
    echo "[INFO] Разрешение sudo для группы wheel активировано."
else
    echo "[INFO] Разрешение sudo для группы wheel уже существует."
fi

# 0.2. создаём группу guest, если её нет
if ! getent group guest >/dev/null ; then
    groupadd guest
    echo "[INFO] Группа guest создана."
fi

wheel_users=() # для итогового вывода
guest_users=()

# -----
# Шаг 1. Интерактивный цикл по созданию пользователей
# -----
while true; do
    read -r -p "Добавить пользователя? [д/у | н/н]: " ans
    ans=$(tr '[:upper:]' '[:lower:]' <<<"$ans")
    [[ $ans =~ ^(н|н)$ ]] && break
    [[ $ans =~ ^(д|у)$ ]] || { echo "Введите 'д' или 'н'"; continue; }

    # 1. выбор роли
    read -r -p "Тип пользователя: wheel(w) / guest(g): " role
    role=$(tr '[:upper:]' '[:lower:]' <<<"$role")
    [[ $role =~ ^(w|g)$ ]] || { echo "Введите 'w' или 'g'"; continue; }

    # 2. имя пользователя
    read -r -p "Имя пользователя: " user
    [[ -z $user ]] && { echo "Имя не может быть пустым."; continue; }

    # 3. создание или проверка существования
```

```

if id "$user" &>/dev/null ; then
    echo "[INFO] $user уже существует."
else
    useradd -m -s /bin/bash "$user"
    # установка пароля
    while true; do
        read -rs -p "Пароль для $user: " p1; echo
        read -rs -p "Повторите пароль: " p2; echo
        [[ $p1 == "$p2" && -n $p1 ]] && break
        echo "Пароли не совпали, попробуйте ещё раз."
    done
    echo "$user:$p1" | chpasswd
    unset p1 p2
    echo "[INFO] Пользователь $user создан."
fi

# 4. включаем в нужную группу
case $role in
    w)
        usermod -aG wheel "$user"
        wheel_users+=("$user")
        echo "[INFO] $user добавлен в группу wheel."
        ;;
    g)
        usermod -aG guest "$user"
        guest_users+=("$user")
        echo "[INFO] $user добавлен в группу guest."
        ;;
esac
done

# -----
# 2. Итоговый отчёт: кто создан
# -----
[[ ${#wheel_users[@]} -gt 0 ]] && echo "new wheel-пользователи: ${wheel_users[*]}"
[[ ${#guest_users[@]} -gt 0 ]] && echo "new guest-пользователи: ${guest_users[*]}"
[[ ${#wheel_users[@]} -eq 0 && ${#guest_users[@]} -eq 0 ]] && \
    echo "Новые аккаунты не созданы."

# -----
# 3. Статус: кто существует теперь (уже созданные и только что созданные пользователи)
# -----
groups=("wheel" "guest")

for group in "${groups[@]}; do
    members=$(getent group "$group" | cut -d: -f4)

    if [[ -z "$members" ]]; then
        echo "Группа '$group' не имеет участников."
    else
        echo "Пользователи группы '$group':"
        IFS=' ' read -ra arr <<< "$members"
        for user in "${arr[@]}; do
            echo "    - $user"
        done
    fi
done
echo
done

```

### 4.3 Задача №4 – Работа с пакетным менеджером DNF

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# 04_dnf_setup.sh - автоматизация задания №4 по установке с помощью пакетного менеджера различных утилит
#
# Запустить от root: sudo ./04_dnf_setup.sh

set -e

# -----
# Шаг 0. Требования запустить от root
# -----
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# Шаг 1. Подключаем дополнительные репозитории (если ещё не включены)
# -----
# EPEL 8 - пакет для screen и pwgen
if ! rpm -q oracle-epel-release-el8 >/dev/null 2>&1; then
    echo "==> Устанавливаем метапакет oracle-epel-release-el8 (EPEL 8)..."
    dnf -y install oracle-epel-release-el8
fi

echo "==> Обновляем кэш репозитория..."
dnf makecache -y

# -----
# Шаг 2. Определяем группы пакетов
# -----
editors=(vim-enhanced nano mc screen)

network=(wget curl telnet nmap-ncat tcpdump net-tools bind-utils)

storage=(autofs nfs-utils cloud-utils-growpart lsof sysfsutils sg3-utils)

monitor=(sysstat)

general=(pwgen bc unzip glibc-langpack-ru)

devtools=(git)

# Единый массив для установки
all_pkgs=(
    "${editors[@]}"
    "${network[@]}"
    "${storage[@]}"
    "${monitor[@]}"
    "${general[@]}"
    "${devtools[@]}"
)

# -----
# Шаг 3. Устанавливаем базовые утилиты
# -----
echo "==> Устанавливаем базовые утилиты (${#all_pkgs[@]} пакетов)..."
dnf install -y "${all_pkgs[@]}"
```

```
# -----
# Шаг 4. Устанавливаем PostgreSQL 15
# -----
echo "==> Переключаемся на модуль PostgreSQL 15 и ставим сервер+клиент..."
dnf module reset -y postgresql
dnf module enable -y postgresql:15
dnf install -y postgresql-server postgresql

# -----
# Шаг 5. Проверка, что всё установлено
# -----
missing=()
for pkg in "${all_pkgs[@]} postgresql postgresql-server; do
    if ! rpm -q "$pkg" &>/dev/null; then
        missing+=("$pkg")
    fi
done

if (($#missing)); then
    echo "Не удалось установить следующие пакеты: ${missing[*]} " >&2
    exit 2
else
    echo "Проверка пройдена успешно: все пакеты на месте."
fi

# -----
# Шаг 6. Итоговый отчёт, если все пакеты на месте
# -----
echo -e "\n===== Итоговая сводка ====="
echo "Редакторы и управление сессиями:"
printf '  • %s\n' "${editors[@]}"

echo -e "\nСетевые клиенты и диагностика:"
printf '  • %s\n' "${network[@]}"

echo -e "\nФайловые системы и хранение:"
printf '  • %s\n' "${storage[@]}"

echo -e "\nСистемный мониторинг и диагностика:"
printf '  • %s\n' "${monitor[@]}"

echo -e "\nУтилиты общего назначения:"
printf '  • %s\n' "${general[@]}"

echo -e "\nРазработка и контроль версий:"
printf '  • %s\n' "${devtools[@]}"

echo -e "\nУстановлен модуль PostgreSQL 15:"
echo '  • postgresql-server (15)'
echo '  • postgresql (клиент 15)'
echo "=====
echo "Готово."
```

## 4.4 Задача №5 – Инициализация и настройка PostgreSQL

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# 05_init_sql.sh - автоматизация задания №5 (Инициализация и настройка PostgreSQL)
# 1. создать базу данных PostgreSQL;
# 2. инициализировать кластер с включёнными контрольными суммами;
# 3. настроить удалённое подключение (поправить файлы postgresql.conf, pg_hba.conf);
#
# Запускать от root: sudo ./05_init_sql.sh

set -e

# -----
# Шаг 0. Требования запустить от root
# -----
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# Шаг 1. Переменные
# -----
PG_VERSION="15"                # версия PostgreSQL
PG_USER="postgres"             # системный пользователь
                                # имя, под которым работает служба PostgreSQL
PG_DATA="/var/lib/pgsql/data"  # каталог кластера, где PostgreSQL хранит данные
PG_DELETE_BACKUP=7             # удалить бэкапы старше N дней

# -----
# Шаг 2. Поиск initdb и имени службы
# -----
if command -v initdb >>/dev/null; then
    PG_BIN="$(dirname "$(command -v initdb)")"
else
    # пробуем типичный путь PGDG
    PG_BIN="/usr/pgsql-`${PG_VERSION}`/bin"
    [[ -x "${PG_BIN}/initdb" ]] \
        || { echo "initdb не найден. Установите пакет postgresql-`${PG_VERSION}`-server."; exit 1; }
fi

if systemctl list-unit-files | grep -q "^postgresql-`${PG_VERSION}`.service"; then
    PG_SERVICE="postgresql-`${PG_VERSION}`"
elif systemctl list-unit-files | grep -q "^postgresql.service"; then
    PG_SERVICE="postgresql"
else
    echo "systemd-служба PostgreSQL не найдена. Убедитесь, что серверный пакет установлен."
    exit 1
fi

echo "[1/5] initdb найден: ${PG_BIN}/initdb"
echo "[1/5] Служба PostgreSQL: ${PG_SERVICE}"

# -----
# Шаг 3. Инициализация кластера (с контрольными суммами)
# -----
echo "[2/5] Инициализация кластера (с контрольными суммами)..."
if [[ -d "${PG_DATA}" && -f "${PG_DATA}/PG_VERSION" ]]; then
    echo "Кластер уже существует, initdb пропущен."
```

```
else
    mkdir -p "${PG_DATA}"
    chmod 700 "${PG_DATA}"
    chown -R "${PG_USER}:${PG_USER}" "${PG_DATA}"
    sudo -u "${PG_USER}" "${PG_BIN}/initdb" --data-checksums -D "${PG_DATA}"
fi

# -----
# Шаг 4. Бэкап конфигов
# -----
CONF="${PG_DATA}/postgresql.conf"
HBA="${PG_DATA}/pg_hba.conf"

echo "[3/5] Бэкап конфигов..."
TIMESTAMP=$(date +%Y%m%d%H%M%S) # чтобы хранить много предыдущих версий
cp "${CONF}" "${CONF}.bak.${TIMESTAMP}"
cp "${HBA}" "${HBA}.bak.${TIMESTAMP}"

find "${PG_DATA}" -maxdepth 1 -type f -name '*.bak.*' -mtime "+${PG_DELETE_BACKUP}" -delete

# -----
# Шаг 5. Настройка конфигов
# -----
echo "[4/5] Настройка конфигов..."

# • postgresql.conf - любые IP, стандартный порт
sed -ri "s/^[#]?listen_addresses\s*=.*\/listen_addresses = '.*'/" "${CONF}"
sed -ri "s/^[#]?port\s*=.*\/port = 5432/" "${CONF}"

# • pg_hba.conf - IPv4 и IPv6, авторизация md5
grep -q "0.0.0.0/0" "${HBA}" || {
    {
        echo ''
        echo 'host all all 0.0.0.0/0 md5'
        echo 'host all all ::/0      md5'
    } >> "${HBA}"
}

# -----
# Шаг 6. Включение автозапуска и запуск службы
# -----
echo "[5/5] Запуск и enable службы..."
systemctl enable "${PG_SERVICE}"
systemctl restart "${PG_SERVICE}"

echo "Готово! Проверьте подключение: psql -h <IP> -U postgres -d postgres"
```



## 4.5 Задача №6 – Развёртывание Демо-базы и настройка доступа

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# 06_demo.sh - развёртывание Демо-БД PostgresPro
#
# 1. Включаем / запускаем службу postgresql
# 2. Создаём (если нет) базу demo_db и роль demo_user
# 3. Скачиваем выбранный архив (small/medium/big) и распаковываем .sql
# 4. Импортируем дампы в demo_db
#
# Запустить от root: sudo ./06_demo.sh

set -e

# -----
# 0. проверка root
# -----
if (( EUID != 0 )); then
    echo "Запустите скрипт от root (sudo)." >&2
    exit 1
fi

# -----
# 1. переменные
# -----
BASE_URL="https://edu.postgrespro.ru"
ARCHIVE_URL="${BASE_URL}/demo-small.zip"          # фиксированная small-версия
TMPDIR="$(mktemp -d)"
ZIP_FILE="${TMPDIR}/demo-small.zip"

DB_NAME="demo_db"
DB_USER="demo_user"
DB_PASS="pass"

# -----
# 2. запускаем службу postgresql
# -----
echo "[1/4] Включаем и запускаем службу postgresql"
systemctl enable --now postgresql

# переходим в нейтральную директорию, чтобы psql не ругался на cwd
cd /tmp

# -----
# 3. создаём базу и роль (идемпотентно)
# -----
echo "[2/4] Создаём (если нужно) базу и роль"

if ! sudo -u postgres psql -tAc "SELECT 1 FROM pg_database WHERE datname='${DB_NAME}'" | grep -q 1; then
    sudo -u postgres psql -c "CREATE DATABASE ${DB_NAME};"
fi

if ! sudo -u postgres psql -tAc "SELECT 1 FROM pg_roles WHERE rolname='${DB_USER}'" | grep -q 1; then
    sudo -u postgres psql -c "CREATE USER ${DB_USER} WITH PASSWORD '${DB_PASS}';"
fi

sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE ${DB_NAME} TO ${DB_USER};"

# -----
# 4. скачиваем Демо-базы и распаковываем
```

```
# -----
echo "[3/4] Скачиваем и распаковываем демо-дамп"
curl -L "${ARCHIVE_URL}" -o "${ZIP_FILE}"
unzip -o "${ZIP_FILE}" -d "${TMPDIR}"

SQL_FILE=$(find "${TMPDIR}" -type f -name '*.sql' | head -n1)
[[ -f "${SQL_FILE}" ]] || { echo "Ошибка: SQL-дамп не найден"; exit 1; }

# делаем доступным для postgres
chown -R postgres:postgres "${TMPDIR}"
chmod -R 750 "${TMPDIR}"

# -----
# 5. импорт в demo_db (импорт через SQL-дамп)
# -----
echo "[4/4] Импортируем данные в ${DB_NAME}"

# удаляем все упоминания базы demo (CREATE/DROP/ALTER/COMMENT) и \connect
sed -e '/^CREATE DATABASE/d' \
    -e '/^DROP DATABASE/d' \
    -e '/^ALTER DATABASE/d' \
    -e '/^COMMENT ON DATABASE/d' \
    -e '/^\\connect demo/d' \
    "${SQL_FILE}" | sudo -u postgres psql -d "${DB_NAME}"

echo "Готово: «${DB_NAME}» наполнена, роль «${DB_USER}» имеет доступ."

rm -rf "${TMPDIR}"
```

#### 4.5.1 Дополнительно - Скрипт для создания новых пользователей в PostgreSQL и настройка прав доступа для них

```
#!/usr/bin/env bash
#
# 06_demo_new_user.sh - добавление новых пользователей в PostgreSQL и настройка прав доступа для него
#
# ПРАВА, КОТОРЫЕ ВЫДАЮТСЯ В ЭТОМ СКРИПТЕ:
#
# GRANT USAGE ON SCHEMA <schema>
# - разрешает пользователю видеть и использовать указанную схему:
#   заходить в неё, писать имена объектов (например, bookings.flights).m.
#
# GRANT CREATE ON SCHEMA <schema>
# - даёт право создавать новые объекты в схеме:
#   таблицы, представления, функции и тд.
#
# GRANT SELECT,INSERT,UPDATE,DELETE ON ALL TABLES IN SCHEMA <schema>
# - стандартный набор CRUD (Create, Read, Update, Delete)
#   привилегий на все существующие таблицы схемы:
#   • SELECT - читать данные (SELECT * FROM ...)
#   • INSERT - добавлять строки (INSERT INTO ...)
#   • UPDATE - изменять строки (UPDATE ... SET ... WHERE ...)
#   • DELETE - удалять строки (DELETE FROM ... WHERE ...)
#
# GRANT SELECT,UPDATE,USAGE ON ALL SEQUENCES IN SCHEMA <schema>
# - права на все последовательности схемы (обычно используются для serial-полей):
#   • USAGE - брать следующее значение (nextval())
#   • SELECT - смотреть текущее значение (currval())
#   • UPDATE - устанавливать или сбрасывать значение (setval())
#
# ALTER DEFAULT PRIVILEGES IN SCHEMA <schema> ...
# - гарантирует, что все новые таблицы и последовательности, создаваемые в схеме,
#   будут доступны уже существующим пользователям
#
# Запускать от root: sudo ./06_demo_new_user.sh

set -e

# -----
# 0. проверка root
# -----
if (( EUID != 0 )); then
    echo "Запустите скрипт от root (sudo)." >&2
    exit 1
fi

# -----
# 1. Параметры
# -----
read -rp "Введите имя базы (по умолчанию demo_db): " DB_NAME
DB_NAME=${DB_NAME:-demo_db}

read -rp "Схема, к которой выдаём права (по умолчанию bookings): " SCHEMA_NAME
SCHEMA_NAME=${SCHEMA_NAME:-bookings}

# набор прав по умолчанию: SELECT, INSERT, UPDATE, DELETE
DEFAULT_PRIVS="SELECT,INSERT,UPDATE,DELETE"
read -rp "Список привилегий для таблиц (ENTER = ${DEFAULT_PRIVS}): " PRIVS
PRIVS=${PRIVS:-$DEFAULT_PRIVS}

# для последовательностей допустим только эти права
```

```

SEQ_PRIVS="SELECT,UPDATE,USAGE"

echo
echo "Работаем с базой:      $DB_NAME"
echo "Схема:                  $SCHEMA_NAME"
echo "Привилегии TABLES:    $PRIVS"
echo "Привилегии SEQUENCES:  $SEQ_PRIVS"
echo

# -----
# 2. Интерактивный цикл (создание пользователя и выдача привелегий)
# -----
while true; do
  read -rp "Введите имя нового пользователя (ENTER = выход): " DB_USER
  [[ -z $DB_USER ]] && { echo "Выход."; break; }

  read -srp "Введите пароль для ${DB_USER}: " DB_PASS
  echo

  # 2.1 создаём пользователя (если нет)
  if psql -U postgres -d "$DB_NAME" -tAc "SELECT 1 FROM pg_roles WHERE rolname='${DB_USER}'" | grep -q
    echo "→ Пользователь '${DB_USER}' уже существует - пропускаю CREATE."
  else
    echo "→ Создаю роль '${DB_USER}'"
    psql -U postgres -d "$DB_NAME" -c "CREATE USER ${DB_USER} WITH PASSWORD '${DB_PASS}';"
  fi

  # 2.2 выдаём права (если ещё не выдавались)
  HAS_USAGE=$(psql -U postgres -d "$DB_NAME" -tAc \
    "SELECT 1 FROM information_schema.role_table_grants
    WHERE grantee='${DB_USER}' LIMIT 1")
  if [[ -z $HAS_USAGE ]]; then
    echo "→ Выдаю права на схему и объекты"
    psql -U postgres -d "$DB_NAME" <<-EOSQL
      GRANT USAGE,CREATE ON SCHEMA ${SCHEMA_NAME} TO ${DB_USER};
      GRANT ${PRIVS}      ON ALL TABLES      IN SCHEMA ${SCHEMA_NAME} TO ${DB_USER};
      GRANT ${SEQ_PRIVS} ON ALL SEQUENCES    IN SCHEMA ${SCHEMA_NAME} TO ${DB_USER};
      -- будущие объекты
      ALTER DEFAULT PRIVILEGES IN SCHEMA ${SCHEMA_NAME}
        GRANT ${PRIVS}      ON TABLES      TO ${DB_USER};
      ALTER DEFAULT PRIVILEGES IN SCHEMA ${SCHEMA_NAME}
        GRANT ${SEQ_PRIVS} ON SEQUENCES TO ${DB_USER};
    EOSQL
  else
    echo "→ Права для '${DB_USER}' уже выдавались - пропускаю GRANT."
  fi

  echo "Готово для пользователя '${DB_USER}'."
  echo "-----"
done

```

## 4.6 Задача №7 – Установка и работа в PgAdmin

### 4.6.1 SQL-запрос, который считает количество вылетов из каждого аэропорта в заданном городе за июль.

Вопрос: «Сколько рейсов вылетело из Москвы в июле и как они распределились по аэропортам?»

```
SELECT
    ap.city,
    ap.airport_name,
    ap.airport_code,
    COUNT(f.flight_id) AS flights_count
FROM
    airports ap
LEFT OUTER JOIN
    flights f
    ON ap.airport_code = f.departure_airport
WHERE
    DATE_PART('month', f.scheduled_departure) = 7
    AND ap.city = 'Москва'
GROUP BY
    ap.city,
    ap.airport_name,
    ap.airport_code;
```

### 4.6.2 SQL-запрос подсчитывает число вылетов из каждого аэропорта в указанном городе за июль и выводит вместе с ним сведения об аэропортах.

Вопрос: «Какие модели самолётов приносят больше всего премиального дохода (наибольшее число бизнес-кресел)?»

```
SELECT DISTINCT
    ac.model,
    (
        SELECT COUNT(s.seat_no)
        FROM aircrafts ac2
        JOIN seats s
            ON ac2.aircraft_code = s.aircraft_code
        WHERE ac2.aircraft_code = ac.aircraft_code
            AND s.fare_conditions = 'Business'
    ) AS business_seat_count
FROM aircrafts ac
ORDER BY
    (
        SELECT COUNT(s.seat_no)
        FROM aircrafts ac2
        JOIN seats s
            ON ac2.aircraft_code = s.aircraft_code
        WHERE ac2.aircraft_code = ac.aircraft_code
            AND s.fare_conditions = 'Business'
    ) DESC;
```

## 5 Выполнение задач – Использование скриптов (дополнительные комментарии)

### 5.1 Задача №2 – Настройка дисковой подсистемы с помощью менеджера томов LVM

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
#
# lvm_setup.sh - автоматизация задания «LVM для PostgreSQL +-расширение root +-snapshot demo»
#
# Запустить от root: sudo ./add_wheel_users.sh
#
# ПЕРЕД СТАРТОМ: при необходимости поправьте переменные в секции «НАСТРОЙКИ»

set -e

#-----
# НАСТРОЙКИ (редактируйте)
#-----
DISK="/dev/sdb"          # дополнительный диск под PostgreSQL
VG="ol"                  # существующая Volume Group (root расположена здесь)
PGLV_NAME="pglv"         # имя логического тома под /var/lib/pgsql
PGLV_SIZE="8G"           # размер тома под PostgreSQL
PG_MOUNT="/var/lib/pgsql"

ROOT_LV="root"           # имя корневого LV внутри $VG
ROOT_EXPAND="+1G"        # сколько добавить к корню

SNAP_SIZE="0.8G"         # объём CoW-области снапшота
TMP_FILE_MB=500          # сколько «мусора» писать для демонстрации (в МБ)
#-----

#-----
# Шаг 0. Требования запустить от root
#-----
# Проверяем, запущен ли скрипт с привилегиями суперпользователя (sudo)
# $EUID - переменная Bash, содержит UID текущего процесса
# $EUID - вернет 0, если script запущен от sudo
# -ne 0 - оператор "not equal". Проверяет, что значение слева не равно 0
# [[ ... ]] - внутри выражение проверяется на истинность. Код = 0, если истина. Код = 1, если ложь
# [[ $EUID -ne 0 ]] - истина, если текущий UID не 0 (то есть не запущено под sudo)
# если не равно 0, то выведется сообщение и script завершится с кодом 1
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

#-----
# ШАГ 1. PostgreSQL на отдельном LV
#-----
setup_pg_lv() {
    echo "=== ШАГ 1. Настройка тома под PostgreSQL ==="

    # 1. Инициализация физического тома (PV) на диске, если не сделано
    # pvs - показывает список PV
    # grep -q - тихий режим, проверяем, есть ли строка, начинающаяся с имени диска
    if ! pvs | grep -q "^${DISK}"; then
        # pvcreate - инициализирует указанный диск как LVM PV
        pvcreate "${DISK}"
    else
```

```

    echo "Пропускаю pvcreate: диск ${DISK} уже инициализирован как PV."
fi

# 2. Добавление PV в Volume Group (VG), если диск ещё не добавлен
#   vgdisplay - показывает параметры VG
#   grep -q   - проверяем, упоминается ли диск в выводе vgdisplay
if ! vgdisplay "${VG}" | grep -q "${DISK}"; then
    # vextend - расширяет существующую VG, добавляя новый PV
    vextend "${VG}" "${DISK}"
else
    echo "Пропускаю vextend: диск уже входит в VG ${VG}."
fi

# 3. Создание логического тома (LV) для PostgreSQL, если он ещё не существует
#   lvs      - показывает список LV
if ! lvs "${VG}/${PGLV_NAME}" &>/dev/null; then
    # lvcreate - создаёт LV размером PGLV_SIZE с именем PGLV_NAME в VG
    lvcreate -L "${PGLV_SIZE}" -n "${PGLV_NAME}" "${VG}"
else
    echo "Пропускаю lvcreate: LV ${PGLV_NAME} уже существует."
fi

# 4. Форматирование LV в файловую систему XFS, если не отформатировано
#   blkid    - проверяет наличие ФС на устройстве
if ! blkid "/dev/${VG}/${PGLV_NAME}" &>/dev/null; then
    # mkfs.xfs - создаёт XFS ФС, -f принудительно перезаписывает старую
    mkfs.xfs -f "/dev/${VG}/${PGLV_NAME}"
else
    echo "Пропускаю mkfs: файловая система уже есть на /dev/${VG}/${PGLV_NAME}."
fi

# 5. Создание точки монтирования
mkdir -p "${PG_MOUNT}"

# 6. Добавление записи в /etc/fstab для автозагрузки file system (FS)
fstab_entry="/dev/${VG}/${PGLV_NAME} ${PG_MOUNT} xfs defaults 0 2"
#   grep -Fxq - точное сравнение строк, тихий режим
if ! grep -Fxq "${fstab_entry}" /etc/fstab; then
    echo "${fstab_entry}" >> /etc/fstab
else
    echo "Пропускаю запись в fstab: запись уже существует."
fi

# 7. Монтируем все FS из fstab
mount -a
echo "Том под PostgreSQL примонтирован в ${PG_MOUNT}"
}

# -----
# ШАГ 2. Расширение корневого раздела
# -----
expand_root() {
    echo "=== ШАГ 2. Увеличение корневого LV на ${ROOT_EXPAND} ==="

    # Команда lvextend:
    #   -r              - одновременно расширить ФС (resize) после изменения размера LV
    #   -L "${ROOT_EXPAND}" - добавить к текущему размеру указанное значение (например, +1G)
    #   "/dev/${VG}/${ROOT_LV}" - путь к логическому тому (VG имя группы, ROOT_LV имя тома)
    lvextend -r -L "${ROOT_EXPAND}" "/dev/${VG}/${ROOT_LV}"
    echo "Корневой LV расширен"
}

```

```

#
# ШАГ 3. Снапшот -> изменения -> откат
#
snapshot_demo() {
    echo "=== ШАГ 3. Демонстрация snapshot ==="

    # 3.1. Удаляем старый снапшот, если он остался от предыдущего запуска
    #     lvs "${VG}/rootsnap" - проверяет существование LV rootsnap
    #     &>/dev/null          - скрывает вывод, важен только код возврата
    #     lvremove -f          - принудительно удаляет логический том
    if lvs "${VG}/rootsnap" &>/dev/null; then
        echo "Удаляю старый снапшот rootsnap"
        lvremove -f "/dev/${VG}/rootsnap"
    fi

    # 3.2. Создаём новый снапшот корневого тома
    #     -L "${SNAP_SIZE}" - размер CoW-области снапшота
    #     -s                - флаг snapshot
    #     -n rootsnap       - имя создаваемого снапшота
    lvcreate -L "${SNAP_SIZE}" -s -n rootsnap "/dev/${VG}/${ROOT_LV}"

    # 3.3. Монтируем снапшот в режиме только для чтения
    mkdir -p /mnt/rootsnap
    #     -o ro              - монтировать в режиме only-read
    #     -o nouuid          - не монтировать UUID, чтобы избежать конфликтов
    mount -o ro,nouuid "/dev/${VG}/rootsnap" /mnt/rootsnap
    echo "Снапшот примонтирован в /mnt/rootsnap (только чтение)"

    # 3.4. Генерируем большие данные для демонстрации изменений
    echo "Создаю ${TMP_FILE_MB} МБ данных в /var/tmp для имитации изменений"
    #     dd if=/dev/zero    - читаем нули
    #     of=/var/tmp/bigfile - записываем в файл
    #     bs=1M count=...    - размер блока 1 МБ, количество блоков TMP_FILE_MB
    #     status=progress    - показывать прогресс
    dd if=/dev/zero of=/var/tmp/bigfile bs=1M count="${TMP_FILE_MB}" status=progress
    sync # ожидаем запись данных на диск

    # 3.5. Показать заполненность CoW-области снапшота
    echo "Заполняемость CoW-области:"
    #     lvs -o ...         - выводим колонки: имя LV, размер, процент данных в CoW, источник (origin)
    lvs -o lv_name,lv_size,data_percent,origin "/dev/${VG}/rootsnap"

    # 3.6. Отмонтировать снапшот перед откатом
    umount /mnt/rootsnap

    # 3.7. Откат к состоянию снапшота
    echo "Выполняю откат к снапшоту (lvconvert --merge)"
    #     lvconvert --merge - объединяет снапшот с исходным томом, возвращая прежнее состояние
    lvconvert --merge "/dev/${VG}/rootsnap"
    echo
    echo "Внимание! Для завершения merge требуется перезагрузка."

    # 3.8. Предложить перезагрузку для применения merge
    read -r -p "Перезагрузить систему сейчас? [y/N] " ans
    # ${ans,,} - приводим ввод к нижнему регистру
    if [[ "${ans,,}" == "y" ]]; then
        reboot
    else
        echo "Перезагрузку можно выполнить позже командой reboot"
    fi
}

```



```
#  
# ГЛАВНЫЙ БЛОК  
#  
setup_pg_lv  
expand_root  
snapshot_demo  
  
echo "Сценарий завершён успешно"
```

## 5.2 Задача №3 – Организация доступа (доп. комментарии)

Реализация задания с помощью скрипта.

```
#!/usr/bin/env bash
# (гарантирует, что скрипт запускается под bash, который будет найден через env в $PATH)
#
# user_setup.sh - автоматизация задания «LVM для PostgreSQL +-расширение root +-snapshot demo»
# 1. разрешение sudo-правила для группы wheel
# 2. создание группы guest
# 3. создание пользователей в цикле, добавляя их в выбранную группу: wheel или guest
#
# !Запускать от root: sudo ./add_wheel_users.sh

set -e # при любой ошибке, т.е. нулевой код возврата, скрипт будет прерван

# -----
# Шаг -1. Требования запустить от root
# -----
# Проверяем, запущен ли скрипт с привилегиями суперпользователя (sudo)
# $EUID - переменная Bash, содержит UID текущего процесса
# $EUID - вернет 0, если script запущен от sudo
# -ne 0 - оператор "not equal". Проверяет, что значение слева не равно 0
# [[ ... ]] - внутри выражение проверяется на истинность. Код = 0, если истина. Код = 1, если ложь
# [[ $EUID -ne 0 ]] - истина, если текущий UID не 0 (то есть не запущено под sudo)
# если не равно 0, то выведется сообщение и script завершится с кодом 1
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# Шаг 0. Подготовка
# -----
# 0.1. включаем правило для wheel в sudoers (с паролем)
# grep:
# grep ищет в файле /etc/sudoers строку, соответствующую регулярному выражению
# grep производит поиск, вызывается с ключами:
# -E включить расширенный синтаксис регулярных выражений
# -q "quiet", т.е. ничего не выводит кроме кода возврата (0 - не найдено, 1 - найдено)
#
# Рег. вып. '^[:space:]*%wheel[:space:]+ALL' :
# ' ' - в кавычки включается регулярное выражение
# Описание:
# 1. ^ - якорь "начало строки", т.е. совпадения должны начинаться с самого начала строки
# 2. [:space:] - класс символов "пробельный символ", т.е. пробел, табуляция (\t), \n
# [:space:]*, где * - квантификатор (0 или более),
# т.е. нам подходит строка с >= 0 пробельными символами
# 3. %wheel - ищем совпадение literal'ного текста
# 4. [:space:]+, где + - квантификатор (1 или более)
# 5. ALL - ищем совпадение literal'ного текста
# Итого ищем строки наподобие:
# 1. %wheel ALL=(ALL) ALL
# 2. %wheel ALL
#
# if ! grep - заходим в блок then, если НЕ НАШЛИ ПОДХОДЯЩИХ строка
#
# sed -i 's/^[:space:]*\(%wheel[:space:]*ALL\)/\1/' /etc/sudoers
# >> sed -i
# sed - stream editor, утилита для строчной обработки текста
# -i - ключ для внесения изменения в файл, сохраняя измененный файл вместо оригинала
# >> s/<то, что хотим заменить>/<заменяем на это>/
# /<то, что хотим заменить>:
```

```

#      1. ^#[[:space:]]* - найти от начала строки все символы # и "пробельные символы"
#      2. \(%wheel[[:space:]].*ALL\) = \(...\) - захватить всё, что идет дальше
#      /<заменяем на это>:
#      1. /\1/, где \1 - замена, \1 значит вставить текст, который был пойман первой группой
#  >> sed:
#      Находит совпадение с шаблоном и запоминает в «группе 1» нужную часть строки.
#      Заменяет всю найденную подстроку на то, что было в группе 1 (\1), фактически удаляя всё,
#      что было до и после самой группы (в данном случае - убирая # и лишние пробелы перед %wheel)
#
# блок then:
#  >> будет выведено сообщение, что теперь правило активировано
#  >> иначе выведено сообщение, что правило уже было активно
if ! grep -Eq '^[[:space:]]*%wheel[[:space:]]+ALL' /etc/sudoers ; then
    sed -i 's/^[[:space:]]*\( %wheel[[:space:]].*ALL\) /\1/' /etc/sudoers
    echo "[INFO] Разрешение sudo для группы wheel активировано."
else
    echo "[INFO] Разрешение sudo для группы wheel уже существует."
fi

# 0.2. создаём группу guest, если её нет
# if ...
#  >> getent group guest - проверка наличия группы
#  >> >/dev/null - перенаправляет вывод команды (текст) getent в мусор, не влияет на код возврата
#  >> getent возвращает 0, если группа существует. Возвращает 2, если запись не найдена.
#  >> ! инверсия кода возврата
# then
#  >> выполнится, если запись не найдена
#  >> groupadd guest - если в системе нет группы guest, создаётся группа
#  >> выведем информационное сообщение

if ! getent group guest >/dev/null ; then
    groupadd guest
    echo "[INFO] Группа guest создана."
fi

# переменные-массивы, в которые будут помещаться новые созданные пользователи
wheel_users=() # для итогового отчёта
guest_users=()

# -----
# Шаг 1. Интерактивный цикл по созданию пользователе
# Вводим админом по одному имени за раз, выбираем для него роль.
# Происходит:
# 1. Проверка существования учетки
# 2. Создание учетки (с домашней папкой и оболочкой)
# 3. Запрос и установка пароля для учетки
# 4. Добавление учетки в нужную группу (wheel или guest)
# -----
while true; do
    # read - чтение одной строки из стандартного ввода
    # -r - ключ для отключения специальной обработки обратного слэша (\)
    # -p - ключ, чтобы не переходить на новую строку после вывода " теста "
    # " наш текст " - формально называется приглашением
    # ans - имя переменной, в которую будет записан введенный ответ
    #
    # ans=$(tr '[:upper:]' '[:lower:]' <<<"$ans")
    # tr '[:upper:]' '[:lower:]' - замена всех заглавных букв на строчные
    # <<<"$ans" - подача содержимого переменной ans в tr
    #
    # [[ $ans =~ ^(n/n)$ ]] && break - если ответ ans 'n' или 'н', то выход из цикла while
    # && - логическое И, правая часть break будет выполнена, если левая часть вернула 0
    # $ans =~ ^(n/n)$ - проверка, что ans соответствует рег. выр ^(n/n)$

```

```

#      ~ - оператор проверки соответствия регулярному выражению
#      ^ - начало строки
#      (д/у) - группа альтернатив, которые нам удовлетворяют
#      $ - конец строки
#
# [[ $ans =~ ^ (д/у) $ ]] || { echo "Введите 'д' или 'н'"; continue; }
# если ответ ans НЕ 'д' или 'у', то выведем подсказку, что нужно вводить
# начнем цикл создания пользователя заново
read -r -p "Добавить пользователя? [д/у | н/н]: " ans
ans=$(tr '[:upper:]' '[:lower:]' <<<"$ans")
[[ $ans =~ ^ (н|н) $ ]] && break
[[ $ans =~ ^ (д|у) $ ]] || { echo "Введите 'д' или 'н'"; continue; }

# 1. выбор роли
# код аналогичен
read -r -p "Тип пользователя: wheel(w) / guest(g): " role
role=$(tr '[:upper:]' '[:lower:]' <<<"$role")
[[ $role =~ ^ (w|g) $ ]] || { echo "Введите 'w' или 'g'"; continue; }

# 2. имя пользователя
# -z - оператор проверки. Возвращает 0, если длина строки равна 0.
# Если введенное имя пользователя равно 0, то выведем сообщение.
# Остальной код аналогичен
read -r -p "Имя пользователя: " user
[[ -z $user ]] && { echo "Имя не может быть пустым."; continue; }

# 3. создание или проверка существования
# 3.1. Проверка существования пользователя
# id "$user" - проверяет, есть ли пользователь в системе
# &>/dev/null - подавляет любой вывод
# если код возврата 0 -> пользователь есть -> выполняем блок then
# иначе код 0 -> пользователь отсутствует -> переходим в блок else
if id "$user" &>/dev/null ; then
    echo "[INFO] $user уже существует."
else
    # 3.2. Создание пользователя
    # useradd - утилита для добавления учётной записи
    # -m - создать домашнюю директорию /home/$user
    # -s /bin/bash - назначить Bash оболочкой по умолчанию
    # "$user" - имя создаваемого пользователя
    useradd -m -s /bin/bash "$user"
    # 3.2. Установка пароля
    # read -r - не интерпретировать слэши (\)
    # -s - не отображать вводимые символы
    # -p "... - вывести подсказку без переноса строки
    # p1 - переменная для первого ввода пароля
    while true; do
        read -rs -p "Пароль для $user: " p1; echo
        read -rs -p "Повторите пароль: " p2; echo # Повторный ввод пароля для подтверждения
        # Проверка совпадения и ненулевой длины:
        # $p1 == "$p2" - оба ввода совпадают
        # -n $p1 - пароль не пустой
        # если оба истинны -> break (выход из цикла), иначе - сообщение и повтор
        [[ $p1 == "$p2" && -n $p1 ]] && break
        echo "Пароли не совпали, попробуйте ещё раз."
    done

    # 3.4. Установка пароля и очистка
    # echo "$user:$p1" - формируем строку "логин:пароль"
    # | chpasswd - передаём в chpasswd для безопасной установки
    echo "$user:$p1" | chpasswd
    # Удаляем переменные с паролями из окружения

```

```

unset p1 p2
echo "[INFO] Пользователь $user создан."
fi

# 4. включаем в нужную группу
case $role in
    w)
        # usermod -aG wheel "$user"
        # -aG wheel - добавить (-a) пользователя в дополнительную группу wheel (сохранив существую
        # "$user" - логин пользователя для добавления
        usermod -aG wheel "$user"
        # wheel_users+=("$user") - добавить имя пользователя в массив wheel_users для итогового отчёта
        wheel_users+=("$user")
        echo "[INFO] $user добавлен в группу wheel."
        ;;
    g)
        # аналогично, но для группы guest
        usermod -aG guest "$user"
        guest_users+=("$user")
        echo "[INFO] $user добавлен в группу guest."
        ;;
esac
done

# -----
# 2. Итоговый отчёт: кто создан
# -----
# Проверяем, есть ли новые пользователи в массиве wheel_users
# ${#wheel_users[@]} - количество элементов в массиве wheel_users
# -gt 0 - больше 0?
# Если да - выводим список через ${wheel_users[*]} (все элементы)
[[ ${#wheel_users[@]} -gt 0 ]] && echo "new wheel-пользователи: ${wheel_users[*]}"
# Аналогично для массива guest_users
[[ ${#guest_users[@]} -gt 0 ]] && echo "new guest-пользователи: ${guest_users[*]}"
# Если ни в одном массиве нет новых пользователей
# -eq 0 - ровно 0 элементов
# Используем логическое И `&&`: обе проверки должны быть истинны
# Объединяем строки через `||` для читаемости
[[ ${#wheel_users[@]} -eq 0 && ${#guest_users[@]} -eq 0 ]] && \
echo "Новые аккаунты не созданы."

# -----
# 3. Статус: кто существует теперь (уже созданные и только что созданные пользователи)
# -----

# 3.1. Задаём массив групп для проверки
# groups - массив имён групп, которые нужно обработать
groups=("wheel" "guest")

# 3.2. Перебираем каждую группу из массива
for group in "${groups[@]}; do
    # 3.2.1. Получаем список участников группы
    # getent group "$group" - извлекает строку из /etc/group
    # cut -d: -f4 - берёт 4-е поле (members), список через запятую
    members=$(getent group "$group" | cut -d: -f4)

    # 3.3. Проверка: есть ли вообще участники
    # [[ -z "$members" ]] - истина, если переменная members пуста (нет участников)
    if [[ -z "$members" ]]; then
        echo "Группа '$group' не имеет участников."
    else

```

```
# 3.4. Выводим заголовок и каждого пользователя по-отдельности
echo "Пользователи группы '$group':"
# 3.4.1. Разбиваем строку members по запятым в массив arr
# IFS=', ' - временно меняем разделитель на запятую
# read -ra arr <<<... - читаем в массив arr, опуская кавычки
IFS=', ' read -ra arr <<< "$members"
# 3.4.2. Перебираем массив и выводим имена с отступом
for user in "${arr[@]}; do
    echo "    - $user"
done
fi
echo # добавление пустой строки
done
```

### 5.3 Задача №4 – Работа с пакетным менеджером DNF (доп. комментарии)

Реализация задания с помощью скрипта.

```
#!/bin/bash
#
# dnf_setup.sh - автоматизация задания №4 по установке с помощью пакетного менеджера различных утилит
#
# Запустить от root: sudo ./add_wheel_users.sh

# -----
# Шаг 0. Требования запустить от root
# -----
# Проверяем, запущен ли скрипт с привилегиями суперпользователя (sudo)
# $EUID - переменная Bash, содержит UID текущего процесса
# $EUID - вернет 0, если скрипт запущен от sudo
# -ne 0 - оператор "not equal". Проверяет, что значение слева не равно 0
# [[ ... ]] - внутри выражение проверяется на истинность. Код = 0, если истина. Код = 1, если ложь
# [[ $EUID -ne 0 ]] - истина, если текущий UID не 0 (то есть не запущено под sudo)
# если не равно 0, то выведется сообщение и скрипт завершится с кодом 1
if [[ $EUID -ne 0 ]]; then
    echo "Запустите скрипт от root!"
    exit 1
fi

# -----
# Шаг 1. Подключаем дополнительные репозитории (если ещё не включены)
# -----
# EPEL 8 - пакет для screen и pwgen
if ! rpm -q oracle-epel-release-el8 >/dev/null 2>&1; then
    # rpm -q oracle-epel-release-el8
    # - проверяет, установлен ли пакет oracle-epel-release-el8
    # >/dev/null 2>&1
    # - перенаправляет вывод в «чёрную дыру», нам важен только код возврата
    # Код возврата 0 -> пакет есть; любой другой -> нет пакета
    echo "==> Устанавливаем метапакет oracle-epel-release-el8 (EPEL 8)..."
    dnf -y install oracle-epel-release-el8
fi
if ! rpm -q oracle-epel-release-el8 >/dev/null 2>&1; then
    echo "==> Устанавливаем метапакет oracle-epel-release-el8 (EPEL 8)..."
    dnf -y install oracle-epel-release-el8
fi

echo "==> Обновляем кэш репозитория..."
# dnf makecache -y
# - загружает и обновляет метаданные репозитория без запроса подтверждения
dnf makecache -y

# -----
# Шаг 2. Определяем группы пакетов
# -----
# 2.1. Текстовые редакторы и консольные утилиты
editors=(vim-enhanced nano mc screen)

# 2.2. Сетевые утилиты
network=(wget curl telnet nmap-ncat tcpdump net-tools bind-utils)

# 2.3. Инструменты для работы с хранилищем и файловой системой
storage=(autofs nfs-utils cloud-utils-growpart lsof sysfsutils sg3_utils)

# 2.4. Мониторинг системы
monitor=(sysstat)
```

```

# 2.5. Общие утилиты
general=(pwgen bc unzip glibc-langpack-ru)

# 2.6. Инструменты разработки
devtools=(git)

# 2.7. Единый массив для установки всех групп пакетов
# all_pkgs - собирает вместе все элементы предыдущих массивов
all_pkgs=(
    "${editors[@]}"
    "${network[@]}"
    "${storage[@]}"
    "${monitor[@]}"
    "${general[@]}"
    "${devtools[@]}"
)

# -----
# Шаг 3. Устанавливаем базовые утилиты
# -----
# ${#all_pkgs[@]} - количество пакетов в массиве all_pkgs
echo "==> Устанавливаем базовые утилиты (${#all_pkgs[@]} пакетов)..."
# dnf install -y - устанавливает без подтверждения (-y)
# "${all_pkgs[@]}" - разворачивает массив всех нужных пакетов
dnf install -y "${all_pkgs[@]}"

# -----
# Шаг 4. Устанавливаем PostgreSQL 15
# -----
echo "==> Переключаемся на модуль PostgreSQL 15 и ставим сервер+клиент..."
# - отмена любых предыдущих настроек модуля (сброс потоков)
dnf module reset -y postgresql
# - включает поток версии 15 модуля postgresql
dnf module enable -y postgresql:15
# - устанавливает пакет сервера и клиентскую утилиту PostgreSQL
dnf install -y postgresql-server postgresql

# -----
# Шаг 5. Проверка, что всё установлено
# -----
missing=() # массив для хранения имён пакетов, которые не установлены
for pkg in "${all_pkgs[@]}" postgresql postgresql-server; do
    # Проверяем, установлен ли пакет pkg
    # rpm -q "$pkg" - запросит информацию о пакете
    # &>/dev/null - подавляет вывод (stdout и stderr), важен только код возврата
    if ! rpm -q "$pkg" &>/dev/null; then
        # Если код возврата не 0 -> пакет не найден, добавляем его в список missing
        missing+=("$pkg")
    fi
done

# ${#missing[@]} - количество элементов в missing
if (($#missing)); then
    # Если массив не пустой -> выводим сообщение и завершаем скрипт с кодом 2
    echo "Не удалось установить следующие пакеты: ${missing[*]}" >&2
    exit 2
else
    # Иначе: сообщение, что все успешно
    echo "Проверка пройдена успешно: все пакеты на месте."
fi

```



```
# -----
# Шаг 6. Итоговый отчёт, если все пакеты на месте
# -----

echo -e "\n==== Итоговая сводка ====="
echo "Редакторы и управление сессиями:"
printf '  • %s\n' "${editors[@]}"

echo -e "\nСетевые клиенты и диагностика:"
printf '  • %s\n' "${network[@]}"

echo -e "\nФайловые системы и хранение:"
printf '  • %s\n' "${storage[@]}"

echo -e "\nСистемный мониторинг и диагностика:"
printf '  • %s\n' "${monitor[@]}"

echo -e "\nУтилиты общего назначения:"
printf '  • %s\n' "${general[@]}"

echo -e "\nРазработка и контроль версий:"
printf '  • %s\n' "${devtools[@]}"

echo -e "\nУстановлен модуль PostgreSQL 15:"
echo '  • postgresql-server (15)'
echo '  • postgresql (клиент 15)'
echo "=====
echo "Готово."
```

## 6 Дополнительно выполненные задачи

### Задача 4 — установка пакетов (DNF).

- Автоматически подключается сторонний репозиторий **EPEL 8** для пакетов, которых нет в стандартных хранилищах (`screen`, `pwgen`).
- После установки репозитория выполняется `dnf makecache`, чтобы ускорить дальнейшую работу.

Где смотреть: файл `04_dnf_setup.sh`, Шаг 1. Подключаем дополнительные репозитории

### Задача 5 — инициализация и настройка PostgreSQL.

- Перед изменением настроек автоматически создаётся резервная копия файлов `postgresql.conf` и `pg_hba.conf` с меткой времени.
- Скрипт очищает каталог: все бэкапы старше определенного количества дней (изначально 7) удаляются командой  

```
find ... -mtime +N -delete.
```

Где смотреть: файл `05_init_sql.sh`, Шаг 4. Бэкап конфигов

### Задача 6 — управление пользователями PostgreSQL.

- Отдельный интерактивный скрипт позволяет последовательно добавлять любых пользователей, назначать пароли и сразу выдавать права.

Где смотреть: файл `06_demo_new_user.sh` (целиком)