

Задачи администрирования Обновление сервера



Авторские права

© Postgres Professional, 2016–2025

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Нумерация версий и общие замечания
Обновление на дополнительный выпуск
Обновление основной версии

Основные версии (9.6, 10, 11, ...)

- поддержка сообщества в течение 5 лет
- добавляется и изменяется функционал
- новая версия не совместима на двоичном уровне с предыдущей
- для обновления всегда требуются специальные действия

Дополнительные выпуски (16.1, 16.2, ...)

- только исправление ошибок и проблем безопасности
- гарантируется двоичная совместимость
- обычно достаточно установить новые исполняемые файлы

Номер версии PostgreSQL состоит из двух частей: номер основной версии (major release) и номер дополнительного выпуска (minor release). До версии 10 основной номер состоял из двух чисел (9.5, 9.6), затем перешли на одно (10, 11). Номер дополнительного выпуска — последнее число через точку (например, 2 в 16.2).

Дополнительные выпуски служат только и исключительно для исправления ошибок, найденных в основной версии. Для них гарантируется сохранение двоичной совместимости (на одной платформе). Поэтому обновление на следующий дополнительный выпуск делается максимально просто и рекомендуется, как только дополнительный выпуск появляется.

Новая основная версия привносит изменение функционала: какие-то возможности добавляются, изменяются, реже — удаляются. В этом случае двоичная совместимость отсутствует. Если попробовать подключить новую версию исполняемых файлов к старой версии кластера БД, PostgreSQL откажется с ним работать.

Для обновления основной версии требуется предпринимать специальные шаги. Не исключено, что помимо обновления сервера БД, потребуются внесение изменений и в приложение. Мотивацией для перехода на новую версию служит появление новых возможностей и окончание поддержки старой версии (которая распространяется на 5 лет с момента выпуска). Из-за сложности процесса часто пропускают несколько версий, например, переходят с 13 сразу на 16 и т. п.

<https://postgrespro.ru/docs/postgresql/16/upgrading>

«Замечания к выпуску»

необходимо изучить раздел «Миграция» всех промежуточных версий

Проверка обновления на тестовом окружении

заранее обнаружить возможные проблемы

автоматизировать процесс для сокращения времени простоя

Запасной вариант

на случай проблем при обновлении производственного сервера

Независимо от того, какое обновление выполняется, стоит обратить внимание на некоторые моменты.

Всегда следует внимательно ознакомиться с приложением «Замечания к выпуску» (Release Notes) документации. В нем в разделе «Миграция» описаны все несовместимости и нестандартные действия, необходимые при обновлении.

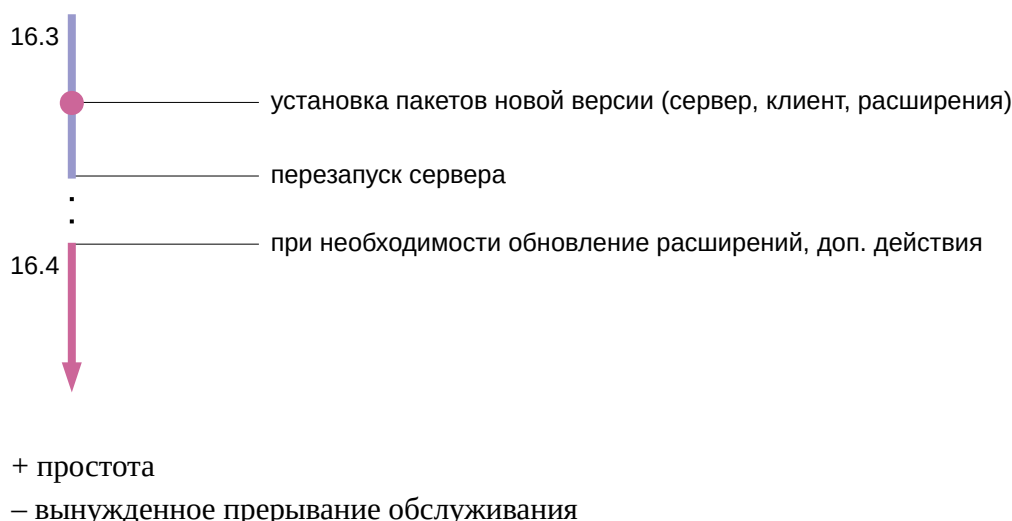
Например, если планируется переход с 15.4 на 16.4 (последний доп. выпуск выбранной основной версии), надо прочитать все замечания к версиям 15.4, ..., 15.8 (последний доп. выпуск для 15), 16.0, ..., 16.4.

Обновление следует проверять на тестовой среде, эквивалентной производственной, с последующим тестированием приложения, что позволит заранее выявить возможные проблемы, а также «обкатать» и автоматизировать процедуру обновления.

Это существенно снижает риск возникновения проблем при обновлении производственного сервера, хотя и не исключает его полностью. Поэтому всегда необходимо иметь (протестированный) запасной вариант на случай, если обновление пойдет не так. Таким вариантом, например, может быть готовая реплика или резервная копия, в зависимости от требований к возможному времени простоя.

Обновление на дополнительный выпуск

Использование физических реплик



Для того чтобы обновить сервер до очередного дополнительного выпуска, требуется:

1. Установить новые исполняемые файлы.

Если PostgreSQL был установлен из пакета, необходимо установить новые версии пакетов (сервер, клиент, расширения). Некоторые пакетные менеджеры, в частности, apt в ОС Ubuntu, по умолчанию автоматически сразу перезапускают сервер; обычно это нежелательно.

Если PostgreSQL был собран из исходных кодов, необходимо выполнить `make install`.

2. Перезапустить сервер. После перезапуска он продолжит работу уже на новой версии. Обслуживание клиентов прерывается на время, требуемое для перезапуска.

3. После перезапуска может понадобиться обновить расширения (`ALTER EXTENSION ... UPDATE`) и выполнить дополнительные действия, указанные в «Замечаниях к выпуску». Однако в большинстве случаев это не требуется.



Обновление средствами операционной системы

```
sudo apt install postgresql-16
```

Может автоматически перезапустить сервер

утилита needrestart

Если PostgreSQL установлен из пакета, операционная система сама может обновить его версию. В Ubuntu для этого используется команда `sudo apt-get install --only-upgrade имя-пакета`

или

```
sudo apt install имя-пакета
```

При таком обновлении сервер обычно автоматически перезапускается, что может быть неудобно. Это поведение настраивается, подробности описаны в документации утилиты needrestart:

<https://github.com/liske/needrestart>

Процесс перезапуска сервера

запрещаются новые соединения

smart ожидает завершения всех сеансов

fast принудительно отключает все сеансы (по умолчанию)

выполняется контрольная точка

Контрольная точка

вручную до перезапуска сервера

PgBouncer

«пауза» на время перезапуска сервера

открытые транзакции продолжают выполняться,
все новые — приостанавливаются

Прерывание обслуживания можно сократить, а в некоторых случаях и устранить полностью.

Что происходит при перезапуске сервера?

Во-первых, PostgreSQL перестает принимать новые подключения (клиенты будут получать ошибку «shutdown in progress»).

Во-вторых, в зависимости от режима останова, сервер либо дожидается завершения активных сеансов, либо принудительно завершает их. Последнее подразумевается по умолчанию утилитой управления, но, вероятно, это не лучший выбор для производственной среды, если сервер в основном имеет дело с OLTP-нагрузкой (короткие запросы).

В-третьих, выполняется финальная контрольная точка (shutdown checkpoint), чтобы синхронизировать данные из буферов в оперативной памяти с диском (как рассматривалось в модуле «Журналирование»). При большом размере буферного кеша этот процесс может занимать значительное время. Поэтому может оказаться целесообразным сначала выполнить контрольную точку вручную (CHECKPOINT), а затем перезапустить сервер. В этом случае финальная контрольная точка все равно будет выполнена, но значительно быстрее.

Если используется PgBouncer (<https://pgbouncer.org>), имеет смысл поставить его на «паузу» на время перезагрузки. При этом (подразумевается режим «transaction») работающие транзакции продолжат выполнение, но все новые будут отложены. Таким образом для OLTP-приложения перезапуск будет выглядеть как увеличившееся время отклика СУБД.



+ возможно обновление без или с минимальным прерыванием обслуживания

9

Если в системе используется физическая репликация и предусмотрен механизм плавного переключения пользователей между серверами, процедуру обновления можно провести и без прерывания обслуживания.

Для этого сначала выполняют обновление резервного сервера. Во время его перезапуска вся нагрузка ложится на основной сервер. Поскольку версии двоично-совместимы, репликация работает между уже обновленным резервным сервером и еще не обновленным основным. В абсолютном большинстве случаев репликация будет работать и в другом направлении (от обновленного сервера к серверу со старой версией), но от ошибок обратной совместимости никто не застрахован. Поэтому сначала лучше обновлять именно резервный сервер.

Затем, когда резервный сервер «догоняет» после перезагрузки основной сервер, основной сервер останавливают и выполняют переключение на резервный. После запуска бывшего основного сервера он подключается к новому основному в качестве резервного (это может потребовать запуска утилиты `pg_rewind`; вся процедура рассматривается подробно в курсе DBA3).

Обновление на дополнительную версию

Выясним текущую версию PostgreSQL:

```
16=> SHOW server_version;

      server_version
-----
16.2 (Ubuntu 16.2-1ubuntu4)
(1 row)
```

А в репозитории есть более новая версия:

```
student$ sudo apt list -q postgresql-16
```

```
postgresql-16/noble-pgdg 16.9-1.pgdg24.04+1 amd64 [upgradable from: 16.2-1ubuntu4]
```

Проведем обновление с версии 16.2 на 16.9 — обновим пакет 'postgresql-16' средствами ОС:

```
student$ sudo apt install -y postgresql-16
```

```
The following additional packages will be installed:
  postgresql-client-16
Suggested packages:
  postgresql-doc-16
The following packages will be upgraded:
  postgresql-16 postgresql-client-16
Preconfiguring packages ...
2 upgraded, 0 newly installed, 0 to remove and 165 not upgraded.
Need to get 0 B/18,3 MB of archives.
After this operation, 17,8 MB of additional disk space will be used.
(Reading database ... 152326 files and directories currently installed.)
Preparing to unpack .../postgresql-client-16_16.9-1.pgdg24.04+1_amd64.deb ...
Unpacking postgresql-client-16 (16.9-1.pgdg24.04+1) over (16.2-1ubuntu4) ...
Preparing to unpack .../postgresql-16_16.9-1.pgdg24.04+1_amd64.deb ...
Unpacking postgresql-16 (16.9-1.pgdg24.04+1) over (16.2-1ubuntu4) ...
Setting up postgresql-client-16 (16.9-1.pgdg24.04+1) ...
Setting up postgresql-16 (16.9-1.pgdg24.04+1) ...
Processing triggers for postgresql-common (278.pgdg24.04+1) ...
Building PostgreSQL dictionaries from installed myspell/hunspell packages...
  en_us
  ru_ru
Removing obsolete dictionary files:
```

Заметим, что пакет postgresql-client-16 тоже обновился.

```
student$ sudo pg_ctlcluster 16 main restart
```

Проверяем номер версии:

```
student$ psql
```

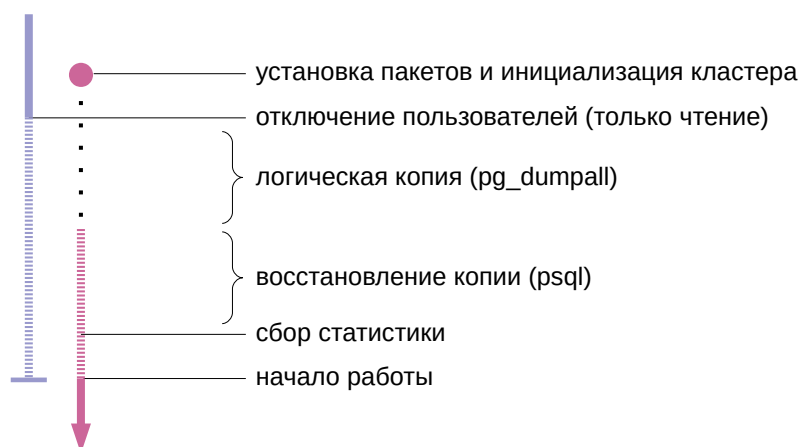
```
16=> SHOW server_version;

      server_version
-----
16.9 (Ubuntu 16.9-1.pgdg24.04+1)
(1 row)
```

Логическая резервная копия `pg_dumpall`

Утилита `pg_upgrade`

Логическая репликация



- + можно сменить платформу, разрядность, кодировку баз данных и т. п.
- большое время обновления, требуется много дискового пространства

Самый простой способ обновления на основную версию — сделать *логическую* резервную копию и развернуть ее на другом сервере с новой версией. (Логическая копия — это набор команд SQL, воссоздающих базу данных; подробнее см. курс DBA3).

Сначала устанавливаем новый сервер и инициализируем кластер (некоторые пакетные менеджеры выполняют инициализацию сразу при установке).

Если новый сервер разворачивается на том же компьютере, надо позаботиться, чтобы каталоги данных не пересекались. Обычно пакеты собираются с учетом этого требования (например, `/var/lib/postgresql/15/` и `/var/lib/postgresql/16/`). Также потребуется изменить конфигурационные файлы нового кластера, внося в них изменения с работающего.

Резервную копию можно делать при работающем сервере, но требуется отключить всех пользователей (кроме только читающих), поскольку изменения, сделанные после запуска резервного копирования, не попадут в копию.

После этого можно запускать новый сервер и в нем разворачивать подготовленную резервную копию. А затем собрать статистику — без этого оптимизатор не сможет строить адекватные планы запросов.

После всех необходимых проверок каталоги старого сервера (исполняемые файлы, содержимое PGDATA, пользовательские табличные пространства) можно удалить.

Ускорение переноса данных

```
pg_dumpall --globals-only
```

```
pg_dump --format=d --jobs=N  
pg_restore --jobs=N
```

 } для каждой базы данных отдельно

Экономия места

```
pg_dumpall | psql параметры-соединения-с-новым-кластером
```

Сбор статистики

```
$ vacuumdb --analyze-in-stages
```

```
расширение dump_stat
```

Быстрая проверка

```
pg_dumpall --schema-only
```

13

Хотя утилита `pg_dumpall` и делает резервную копию всего кластера, она всегда выполняется в один поток. Восстановление также выполняется в один поток утилитой `psql`.

Если аппаратные ресурсы позволяют, можно сохранить глобальные объекты кластера с помощью `pg_dumpall`, а затем выполнить копии каждой БД отдельно в параллельном режиме с помощью утилиты `pg_dump` в формате `directory`. В таком случае и восстановление можно выполнять в параллельном режиме утилитой `pg_restore`. Подробности см. в курсе DBA3.

Если стоит задача сэкономить место на диске, можно не сохранять резервную копию, а просто направить вывод `pg_dumpall` на вход `psql`.

Чтобы как можно раньше собрать хоть какую-то статистику, можно выполнять ее сбор в несколько (3) этапов. Можно попробовать начать работу с СУБД уже после второго этапа, сократив тем самым время прерывания обслуживания. Еще одна возможность — использовать внешнее расширение `dump_stat`:

<https://postgrespro.ru/docs/postgrespro/16/dump-stat>

При проверке обновления на тестовой среде сначала можно выполнить обновление с переносом только схемы данных (без самих данных). Таким образом можно достаточно быстро выявить часть возможных проблем.

Заметим, что лучше использовать все утилиты от новой (уже установленной) версии, поскольку они могут содержать улучшения, недоступные в предыдущей версии.

Кластер PostgreSQL 15

Остановим кластер 16 main, чтобы случайно к нему не подключиться.

```
student$ sudo pg_ctlcluster 16 main stop
```

В каталоге /var/lib/postgresql/15/prod находится кластер баз данных PostgreSQL версии 15.

```
student$ psql -p 5433
```

```
| 15=> SHOW server_version;
|
|          server_version
| -----
| 15.13 (Ubuntu 15.13-1.pgdg24.04+1)
| (1 row)
```

Создадим табличное пространство и базу данных.

```
student$ sudo rm -rf /var/lib/postgresql/ts_dir
```

```
student$ sudo mkdir /var/lib/postgresql/ts_dir
```

```
student$ sudo chown postgres: /var/lib/postgresql/ts_dir
```

```
| 15=> CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir';
|
| CREATE TABLESPACE
|
| 15=> CREATE DATABASE admin_upgrade;
|
| CREATE DATABASE
|
| 15=> \c admin_upgrade
|
| You are now connected to database "admin_upgrade" as user "student".
```

Создадим таблицу в созданном табличном пространстве:

```
| 15=> CREATE TABLE test(
|       id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
|       s text
| ) TABLESPACE ts;
|
| CREATE TABLE
|
| 15=> INSERT INTO test(s) VALUES ('Привет от версии 15!');
|
| INSERT 0 1
```

Установим расширение pgaudit:

```
student$ sudo apt install -y postgresql-15-pgaudit
```

The following NEW packages will be installed:

postgresql-15-pgaudit

0 upgraded, 1 newly installed, 0 to remove and 165 not upgraded.

Need to get 0 B/45,1 kB of archives.

After this operation, 108 kB of additional disk space will be used.

Selecting previously unselected package postgresql-15-pgaudit.

(Reading database ... 152672 files and directories currently installed.)

Preparing to unpack .../postgresql-15-pgaudit_1.7.1-1.pgdg24.04+1_amd64.deb ...

Unpacking postgresql-15-pgaudit (1.7.1-1.pgdg24.04+1) ...

Setting up postgresql-15-pgaudit (1.7.1-1.pgdg24.04+1) ...

Processing triggers for postgresql-common (278.pgdg24.04+1) ...

Building PostgreSQL dictionaries from installed myspell/hunspell packages...

en_us

ru_ru

Removing obsolete dictionary files:

```
| 15=> ALTER SYSTEM SET shared_preload_libraries = 'pgaudit';
|
| ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 15 prod restart
```

```
student$ psql -p 5433
```

```
| 15=> \c admin_upgrade
|
| You are now connected to database "admin_upgrade" as user "student".
```

```
15=> CREATE EXTENSION pgaudit;
```

```
CREATE EXTENSION
```

```
15=> \dx pgaudit
```

```

              List of installed extensions
  Name      | Version | Schema | Description
-----+-----+-----+-----
pgaudit     | 1.7.1   | public | provides auditing functionality
(1 row)
```

Утилита pg_dumpall

Сделаем резервную копию всего кластера. Сервер должен работать, но изменения, сделанные после запуска pg_dumpall, в копию не попадут.

```
student$ pg_dumpall -p 5433 > ~/dump.sql
```

Созданная резервная копия — текстовый файл с командами SQL. В нем есть команды для создания баз данных:

```
student$ grep 'CREATE DATABASE' ~/dump.sql | fold -sw 100
```

```
CREATE DATABASE admin_upgrade WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE_PROVIDER = libc
LOCALE = 'en_US.UTF-8';
CREATE DATABASE student WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE_PROVIDER = libc LOCALE =
'en_US.UTF-8';
```

И команды для создания таблицы:

```
student$ grep -A 3 'CREATE TABLE ' ~/dump.sql
```

```
CREATE TABLE public.test (
  id integer NOT NULL,
  s text
);
```

А также команды для создания таких объектов уровня кластера, как роли и табличные пространства, например:

```
student$ grep 'CREATE TABLESPACE' ~/dump.sql
```

```
CREATE TABLESPACE ts OWNER student LOCATION '/var/lib/postgresql/ts_dir';
```

Мы не будем восстанавливать резервную копию на сервере PostgreSQL версии 16; такое задание есть в практике.

Условия применимости

- в новой версии изменяется формат только служебной информации
- в остальном сохраняется двоичная совместимость, включая представление типов и файлы данных
- в базах данных не используются REG-типы
- обновление с версии 8.4 или более поздней
- обновление только до версии, соответствующей утилите

Второй, наиболее распространенный, способ обновления — утилита `pg_upgrade`, которая умеет приводить файлы кластера к виду, совместимому с новой основной версией.

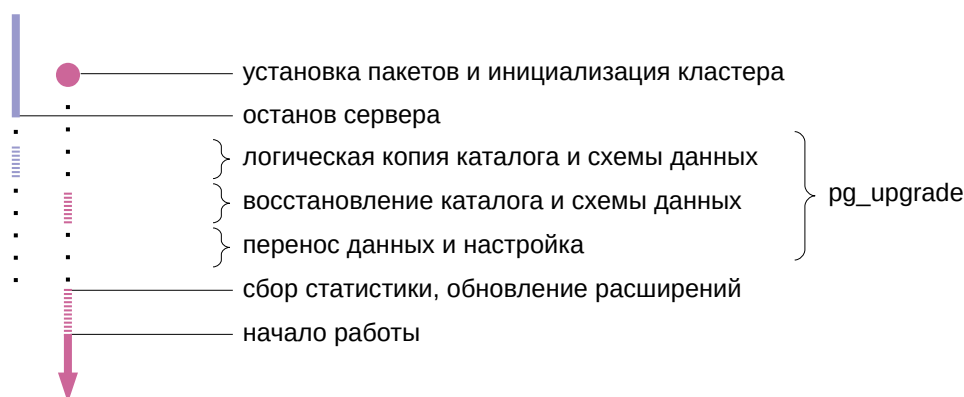
<https://postgrespro.ru/docs/postgresql/16/pgupgrade>

Возможность применения утилиты обусловлена тем, что при обновлении основной версии меняются служебные таблицы, относящиеся к системному каталогу, но форматы файлов данных и индексов как правило остаются без изменений. Поэтому достаточно поправить только небольшую часть данных, оставив основной массив как есть, без изменений. При этом новый кластер должен быть совместим со старым по разрядности, кодировке и локалям.

В некоторых версиях (достаточно редко) изменяется двоичное представление типов данных. В таких случаях утилита не сможет помочь, но по крайней мере перечислит, где используется проблемный тип. Это поможет избавиться от него вручную перед обновлением.

Есть и дополнительные ограничения. Например, утилита не сможет обновить кластер, если в одной из баз данных используются REG-типы (кроме `regtype`, `regclass` и `regrole`).

При запуске `pg_upgrade` проверяет совпадение своей версии и версий исполняемых файлов, задействованных в обновлении, с версией нового сервера. Утилитой поддерживается обновление любой версии, начиная с 8.4, на версию, соответствующую утилите. Таким образом, `pg_upgrade` от версии 16 может обновить старый сервер только до 16, но не до 15. Уменьшение версии сервера (downgrade) невозможно.



+ скорость обновления, обычно не требуется дополнительное место
– ограниченная применимость

16

Вначале рассмотрим общую (упрощенную) последовательность действий при обновлении с помощью `pg_upgrade`.

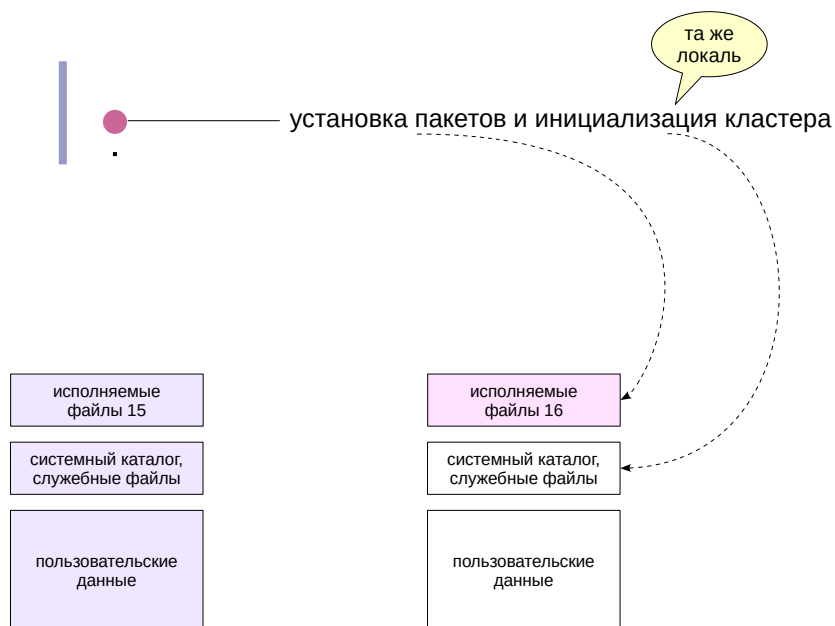
Сначала необходимо установить новый сервер и инициализировать (но не запускать) кластер. Этот шаг выполняется так же, как и при обновлении с помощью `pg_dumpall`.

Затем старый сервер останавливается и запускается утилита `pg_upgrade`. Пока обозначим схематически, что она делает:

- Старый сервер запускается на время, чтобы сделать *логическую* резервную копию общих объектов кластера и схемы данных всех БД. Это как раз та информация, для которой теряется двоичная совместимость.
- Временно запускается новый сервер, выполняются необходимые проверки применимости, из резервной копии восстанавливаются глобальные объекты и схема данных.
- Выполняется перенос данных: либо копирование, либо простановка жестких ссылок, либо клонирование (для файловых систем с копированием при записи, *copy-on-write*).

В результате работы утилиты новый кластер становится обновленной версией старого. Далее выполняются заключительные действия, такие, как сбор статистики и обновление расширений.

Огромный плюс `pg_upgrade` состоит в возможности очень быстрого обновления кластера любого размера без дополнительного места на диске (при использовании жестких ссылок). Из минусов отметим то, что утилита применима не всегда.



Чуть более подробно рассмотрим, что происходит с файловыми системами старого и нового кластера при обновлении.

Шаг 1. Установлены пакеты с новой версией и инициализирован кластер. Новый сервер выключен и не содержит никаких пользовательских объектов, только стандартные базы данных и системный каталог.

Старый сервер продолжает работать.

Важный момент: новый кластер должен быть инициализирован с той же локалью, что и старый.

Кластер PostgreSQL 16

PostgreSQL версии 16 уже установлен, кластер инициализирован в каталоге `/var/lib/postgresql/16/prod`.

Убедимся, что сервер работает и использует ту же локаль, что и PostgreSQL 15, после чего остановим его.

```
15=> \x \l template0

Expanded display is on.
List of databases
-[ RECORD 1 ]-----+
Name           | template0
Owner          | postgres
Encoding       | UTF8
Locale Provider | libc
Collate        | en_US.UTF-8
Ctype          | en_US.UTF-8
ICU Locale     |
ICU Rules      |
Access privileges | =c/postgres          +
                | postgres=CTc/postgres
```

```
15=> \q
```

```
student$ sudo pg_ctlcluster 15 prod stop
```

```
student$ sudo pg_ctlcluster 16 prod start
```

```
student$ psql -p 5433 -U postgres
```

```
16=> \x \l template0

Expanded display is on.
List of databases
-[ RECORD 1 ]-----+
Name           | template0
Owner          | postgres
Encoding       | UTF8
Locale Provider | libc
Collate        | en_US.UTF-8
Ctype          | en_US.UTF-8
ICU Locale     |
ICU Rules      |
Access privileges | =c/postgres          +
                | postgres=CTc/postgres
```

Остановим сервер.

```
16=> \q
```

```
student$ sudo pg_ctlcluster 16 prod stop
```

Проверка возможности обновления

Перед тем как выполнять настоящее обновление, имеет смысл запустить `pg_upgrade` в режиме проверки с ключом `--check`. Мы планируем не копировать файлы данных, а использовать ссылки, поэтому указываем ключ `--link`.

Также мы должны указать пути к исполняемым файлам и к каталогу данных как для старой версии, так и для новой. Обратите внимание, что программа запускается от имени пользователя ОС `postgres`, так как ей требуется доступ к каталогам данных.

Важно, чтобы версии `pg_upgrade` и других серверных утилит (`pg_controldata`, `initdb`) соответствовали версиям клиентских утилит, которые она вызывает (`pg_dump`, `pg_dumpall`, `pg_restore`, `psql`, `vacuumdb`), иначе при запуске получим ошибку. Поскольку мы обновили на версию 16.9 и серверный, и клиентский пакеты, это требование выполнено:

```
postgres$ /usr/lib/postgresql/16/bin/pg_upgrade --version
```

```
pg_upgrade (PostgreSQL) 16.9 (Ubuntu 16.9-1.pgdg24.04+1)
```

```
student$ pg_dump --version
```

```
pg_dump (PostgreSQL) 16.9 (Ubuntu 16.9-1.pgdg24.04+1)
```

Следует обратить внимание на настройки доступа. Утилита в процессе работы запускает и останавливает серверы, и для этого нужно, чтобы к обоим кластерам у нее был локальный суперпользовательский доступ. У нас такой доступ есть, а в

общем случае может потребоваться временно изменить файл pg_hba.conf. Также программа pg_upgrade создает различные временные файлы в специально создаваемом рабочем каталоге pg_upgrade_output.d внутри каталога данных нового кластера, доступ на запись к которому ей тоже необходим. Серверы поднимаются по очереди на порту 50432, чтобы не допустить случайного подключения к ним пользователей; при необходимости номер порта можно указать явно.

```
postgres$ /usr/lib/postgresql/16/bin/pg_upgrade \
--check \
--link \
-b /usr/lib/postgresql/15/bin/ \
-B /usr/lib/postgresql/16/bin/ \
-d /etc/postgresql/15/prod \
-D /etc/postgresql/16/prod

Finding the real data directory for the source cluster      ok
Finding the real data directory for the target cluster     ok
Performing Consistency Checks
-----
Checking cluster versions                                  ok
Checking database user is the install user                 ok
Checking database connection settings                      ok
Checking for prepared transactions                        ok
Checking for system-defined composite types in user tables ok
Checking for reg* data types in user tables               ok
Checking for contrib/iso with bigint-passing mismatch    ok
Checking for incompatible "aclitem" data type in user tables ok
Checking for presence of required libraries                fatal
```

Your installation references loadable libraries that are missing from the new installation. You can add these libraries to the new installation, or remove the functions using them from the old installation. A list of problem libraries is in the file:

```
/var/lib/postgresql/16/prod/pg_upgrade_output.d/20250623T235147.084/loadable_libraries.txt
Failure, exiting
```

Утилита обнаружила проблему: в новом кластере не хватает библиотек.

Посмотрим их список:

```
student$ sudo cat
/var/lib/postgresql/16/prod/pg_upgrade_output.d/20250623T235147.084/loadable_libraries.txt

could not load library "$libdir/pgaudit": ERROR: could not access file
"$libdir/pgaudit": No such file or directory
In database: admin_upgrade
```

Это библиотека установленного нами расширения, от которого зависит индекс в базе admin_upgrade. Ее необходимо установить и в новом кластере.

```
student$ sudo apt install -y postgresql-16-pgaudit
```

```
The following NEW packages will be installed:
  postgresql-16-pgaudit
0 upgraded, 1 newly installed, 0 to remove and 165 not upgraded.
Need to get 0 B/46,2 kB of archives.
After this operation, 108 kB of additional disk space will be used.
Selecting previously unselected package postgresql-16-pgaudit.
(Reading database ... 152683 files and directories currently installed.)
Preparing to unpack .../postgresql-16-pgaudit_16.1-1.pgdg24.04+1_amd64.deb ...
Unpacking postgresql-16-pgaudit (16.1-1.pgdg24.04+1) ...
Setting up postgresql-16-pgaudit (16.1-1.pgdg24.04+1) ...
Processing triggers for postgresql-common (278.pgdg24.04+1) ...
Building PostgreSQL dictionaries from installed myspell/hunspell packages...
  en_us
  ru_ru
Removing obsolete dictionary files:
```

Не забываем дать указание загружать библиотеку в память каждого обслуживающего процесса:

```
student$ sudo pg_ctlcluster 16 prod start

student$ psql -p 5433 -U postgres

16=> ALTER SYSTEM SET shared_preload_libraries = 'pgaudit';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 prod stop
```

Проверяем еще раз.

```
postgres$ /usr/lib/postgresql/16/bin/pg_upgrade \
```

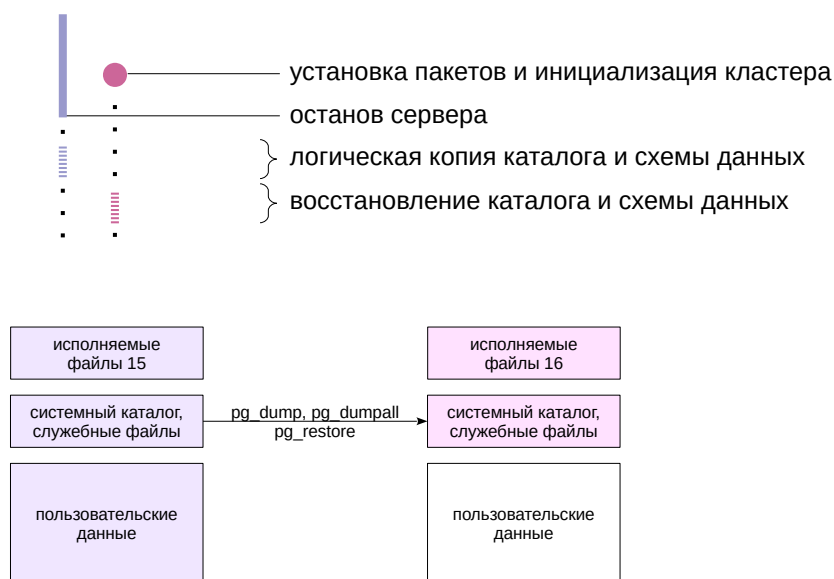
```
--check \  
--link \  
-b /usr/lib/postgresql/15/bin/ \  
-B /usr/lib/postgresql/16/bin/ \  
-d /etc/postgresql/15/prod \  
-D /etc/postgresql/16/prod
```

```
Finding the real data directory for the source cluster      ok  
Finding the real data directory for the target cluster      ok  
Performing Consistency Checks
```

```
-----  
Checking cluster versions                                  ok  
Checking database user is the install user                 ok  
Checking database connection settings                       ok  
Checking for prepared transactions                         ok  
Checking for system-defined composite types in user tables ok  
Checking for reg* data types in user tables                ok  
Checking for contrib/isn with bigint-passing mismatch      ok  
Checking for incompatible "aclitem" data type in user tables ok  
Checking for presence of required libraries                ok  
Checking database user is the install user                 ok  
Checking for prepared transactions                         ok  
Checking for new cluster tablespace directories            ok
```

Clusters are compatible

Теперь кластеры совместимы.



Шаг 2. Старый сервер останавливается, запускается утилита `pg_upgrade`. Она создает резервную копию каталога и схем данных всех баз с помощью утилит `pg_dump` и `pg_dumpall` в особом режиме `--binary-upgrade`, который сохраняет нумерацию файлов (этот режим не нужен для обычной работы и официально не поддерживается):

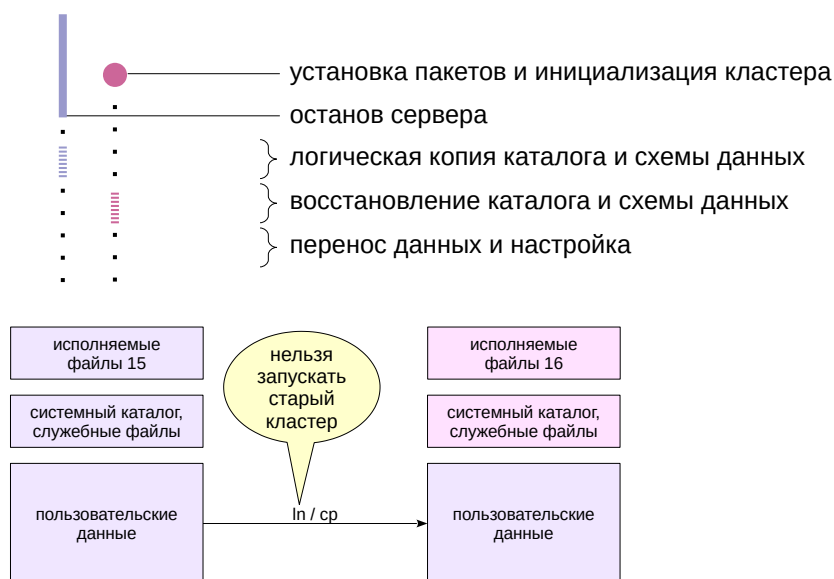
```
pg_dumpall --globals-only --binary-upgrade
pg_dump --schema-only --binary-upgrade
```

Затем эти объекты восстанавливаются на новом сервере с помощью `pg_restore` (подробно эти утилиты рассматриваются в курсе DBA3). Имена файлов каталогов для баз данных и табличных пространств, а также имена файлов отношений остаются прежними после обновления.

Все эти действия выполняются достаточно быстро — время не зависит от объема данных в кластере.

После этого системный каталог нового кластера будет соответствовать системному каталогу старого кластера с точностью до изменения формата данных (именно поэтому для него и используется логическая, а не физическая резервная копия). Фактически, в новом кластере будут созданы все базы данных и все объекты внутри них, но системный каталог будет ссылаться на еще не существующие файлы данных.

На самом деле утилита `pg_upgrade` выполняет еще много довольно тонких операций. Она замораживает все транзакции нового кластера (чтобы не зависеть от номеров транзакций, которые там выполнялись), копирует статусы транзакций (CLOG), копирует информацию о мультитранзакциях и устанавливает счетчик транзакций в значение, взятое со старого кластера.



Шаг 3. Утилита `pg_upgrade` выполняет перенос пользовательских данных.

Теперь либо файлы данных нового кластера скопированы со старого, либо на них проставлены жесткие ссылки.

Заметим, что если файлы данных копируются, то старый кластер никак не затрагивается обновлением и можно в любой момент вернуться к нему, если при обновлении что-то пошло не так.

Однако при использовании жестких ссылок файлы данных фактически становятся общими — точнее, одноименные файлы двух кластеров ссылаются на одинаковые индексные дескрипторы файловой системы (inode). Поэтому, как только новый кластер будет запущен первый раз, старый уже станет невозможно запустить безопасным образом. Такой режим обновления требует наличия «плана Б».

Для файловых систем с `copy-on-write` возможен третий вариант — клонирование (`--clone`), сочетающий преимущества двух предыдущих: быстрый перенос данных и возможность запуска обоих кластеров.

После переноса пользовательских данных утилите остается выставить счетчик `OID` в значение со старого кластера (это выполняется утилитой `pg_resetwal`).

А администратору остается обновить расширения, собрать статистику и не забыть про плановое резервное копирование обновленного кластера.

Обновление

Выполняем обновление в режиме создания ссылок.

```
postgres$ /usr/lib/postgresql/16/bin/pg_upgrade \  
--link \  
-b /usr/lib/postgresql/15/bin/ \  
-B /usr/lib/postgresql/16/bin/ \  
-d /etc/postgresql/15/prod \  
-D /etc/postgresql/16/prod
```

```
Finding the real data directory for the source cluster      ok  
Finding the real data directory for the target cluster     ok  
Performing Consistency Checks  
-----  
Checking cluster versions                                  ok  
Checking database user is the install user                 ok  
Checking database connection settings                      ok  
Checking for prepared transactions                        ok  
Checking for system-defined composite types in user tables ok  
Checking for reg* data types in user tables               ok  
Checking for contrib/isn with bigint-passing mismatch     ok  
Checking for incompatible "aclitem" data type in user tables ok  
Creating dump of global objects                           ok  
Creating dump of database schemas                         ok  
Checking for presence of required libraries               ok  
Checking database user is the install user                 ok  
Checking for prepared transactions                        ok  
Checking for new cluster tablespace directories           ok
```

If pg_upgrade fails after this point, you must re-initdb the new cluster before continuing.

Performing Upgrade

```
-----  
Setting locale and encoding for new cluster                ok  
Analyzing all rows in the new cluster                      ok  
Freezing all rows in the new cluster                      ok  
Deleting files from new pg_xact                           ok  
Copying old pg_xact to new server                         ok  
Setting oldest XID for new cluster                        ok  
Setting next transaction ID and epoch for new cluster     ok  
Deleting files from new pg_multixact/offsets              ok  
Copying old pg_multixact/offsets to new server            ok  
Deleting files from new pg_multixact/members              ok  
Copying old pg_multixact/members to new server            ok  
Setting next multixact ID and offset for new cluster      ok  
Resetting WAL archives                                    ok  
Setting frozenxid and minmxid counters in new cluster     ok  
Restoring global objects in the new cluster                ok  
Restoring database schemas in the new cluster              ok  
Adding ".old" suffix to old global/pg_control             ok
```

If you want to start the old cluster, you will need to remove the ".old" suffix from /var/lib/postgresql/15/prod/global/pg_control.old. Because "link" mode was used, the old cluster cannot be safely started once the new cluster has been started.

```
Linking user relation files                               ok  
Setting next OID for new cluster                          ok  
Sync data directory to disk                              ok  
Creating script to delete old cluster                     ok  
Checking for extension updates                            notice
```

Your installation contains extensions that should be updated with the ALTER EXTENSION command. The file update_extensions.sql when executed by psql by the database superuser will update these extensions.

Upgrade Complete

```
-----  
Optimizer statistics are not transferred by pg_upgrade.  
Once you start the new server, consider running:  
/usr/lib/postgresql/16/bin/vacuumdb --all --analyze-in-stages  
Running this script will delete the old cluster's data files:  
./delete_old_cluster.sh
```


Перед запуском сервера следовало бы перенести изменения, сделанные в конфигурационных файлах старого сервера, на новый. Мы не будем этого делать, поскольку работаем с настройками по умолчанию.

Итак, проверим результат.

```
student$ sudo pg_ctlcluster 16 prod start
```

```
student$ psql -p 5433
```

```
16=> SHOW server_version_num;
```

```
server_version_num
-----
160009
(1 row)
```

```
16=> \c admin_upgrade
```

You are now connected to database "admin_upgrade" as user "student".

```
16=> SELECT * FROM test;
```

```
id | s
---+---
 1 | Привет от версии 15!
(1 row)
```

Обновление прошло успешно: нам доступно содержимое старого кластера.

Табличные пространства

Посмотрим на файлы табличного пространства:

```
student$ sudo tree /var/lib/postgresql/ts_dir --inodes
```

```
[1441833] /var/lib/postgresql/ts_dir
├── [1441851] PG_15_202209061
│   └── [1443470] 16387
│       ├── [1443471] 16389
│       ├── [1443472] 16392
│       └── [1443473] 16393
└── [1443543] PG_16_202307071
    └── [1444744] 16387
        ├── [1443471] 16389
        ├── [1443472] 16392
        └── [1443473] 16393
```

5 directories, 6 files

- Внутри каталога создается подкаталог для каждой версии, поэтому пересечения по файлам не происходит.
- Как видно по числу в квадратных скобках (inode), файлы в каталогах старой и новой версий на самом деле разделяют общее содержимое.

Действия после обновления

Библиотеки установленных расширений заменяются при установке новой версии, но на уровне объектов баз данных версия расширения остается неизменной:

```
16=> \dx pgaudit
```

```
          List of installed extensions
Name | Version | Schema | Description
-----+-----+-----+-----
pgaudit | 1.7.1 | public | provides auditing functionality
(1 row)
```

Утилита обнаружила этот факт и сгенерировала скрипт, обновляющий расширения во всех базах данных:

```
student$ sudo cat /var/lib/postgresql/update_extensions.sql
```

```
\connect admin_upgrade
ALTER EXTENSION "pgaudit" UPDATE;
```

Выполним его.

```
student$ sudo psql -U postgres -p 5433 -f /var/lib/postgresql/update_extensions.sql
```

You are now connected to database "admin_upgrade" as user "postgres".

```
psql:/var/lib/postgresql/update_extensions.sql:2: ERROR:  extension "pgaudit" has no
update path from version "1.7.1" to version "16.1"
```

В нашем случае разработчики расширения не предоставили скрипт для обновления версии, поэтому придется удалить расширение и установить его заново.

```
16=> DROP EXTENSION pgaudit;
```

```
DROP EXTENSION
```

```
16=> CREATE EXTENSION pgaudit;
```

```
CREATE EXTENSION
```

```
16=> \dx pgaudit
```

```
          List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
pgaudit | 16.1    | public | provides auditing functionality
(1 row)
```

Утилита pg_upgrade сгенерировала еще один скрипт, удаляющий файлы старого кластера:

```
student$ sudo cat /var/lib/postgresql/delete_old_cluster.sh
```

```
#!/bin/sh
```

```
rm -rf '/var/lib/postgresql/15/prod'
rm -rf '/var/lib/postgresql/ts_dir/Pg_15_202209061'
```

И напомнила, что нужно собрать статистику. Ключ --analyze-in-stages просит собрать статистику в три этапа, каждый раз увеличивая ее точность, чтобы оптимизатор смог начать строить планы как можно раньше.

```
student$ /usr/lib/postgresql/16/bin/vacuumdb -p 5433 --all --analyze-in-stages
```

```
vacuumdb: processing database "admin_upgrade": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "postgres": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "student": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "template1": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "admin_upgrade": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "postgres": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "student": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "template1": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "admin_upgrade": Generating default (full) optimizer
statistics
vacuumdb: processing database "postgres": Generating default (full) optimizer statistics
vacuumdb: processing database "student": Generating default (full) optimizer statistics
vacuumdb: processing database "template1": Generating default (full) optimizer statistics
```



Утилита-обертка

pg_upgradecluster

pg_dump + pg_restore

--method=dump

pg_upgrade

--method=upgrade

старый кластер переключается на другой порт

файлы конфигурации копируются

pg_dump: можно сменить локаль

pg_upgrade: копирование файлов или жесткие ссылки

Неприменима для табличных пространств

Утилита-обертка pg_upgradecluster для ОС Ubuntu облегчает обновление кластера в простых случаях. По умолчанию используется вариант с логической копией, также поддерживается обновление утилитой pg_upgrade как в режиме копирования файлов, так и в режиме жестких ссылок.

Новый кластер использует тот же порт, а старый настраивается на другой (неиспользуемый) порт и ручной запуск. Утилита копирует конфигурационные файлы в новый кластер, поэтому может потребоваться их дополнительная настройка.

При обновлении через логическую копию можно поменять локаль или отдельные категории локали.

Но если в кластере имеются дополнительные табличные пространства, придется выполнять обновление вручную: утилита такую конфигурацию не поддерживает.

Реплики нужно создать заново

- обновление обязательно (обеспечение двоичной совместимости)
- использование `pg_upgrade` невозможно (недетерминированность)

Вариант 1 (`pg_basebackup`)

- развернуть новые реплики из физической резервной копии, сделанной после обновления основного сервера
- надежно, но долго

Вариант 2 (`rsync`)

- воспользоваться тем, что файлы данных не изменяются
- применимо только для Unix и только для жестких ссылок
- быстро, но сложно

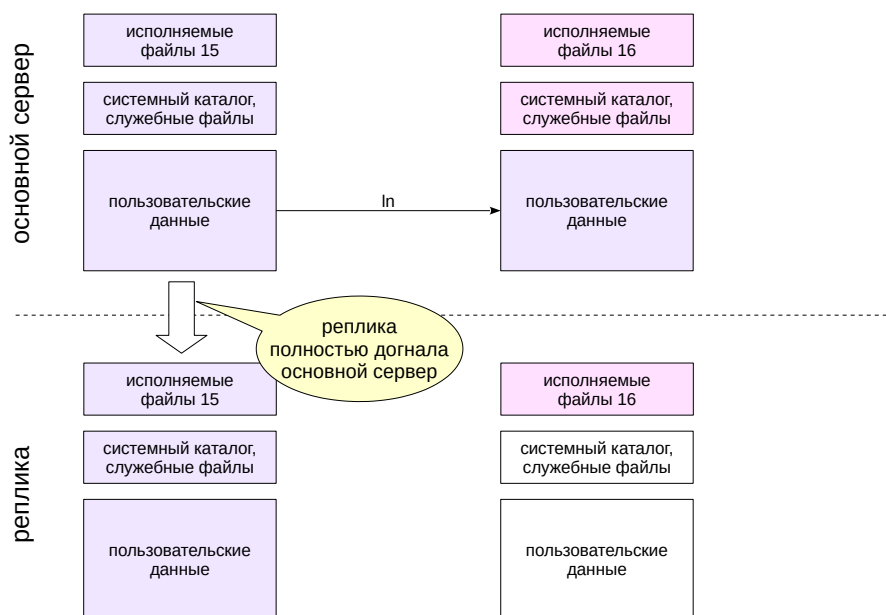
При использовании физической репликации возникают дополнительные трудности. Дело в том, что и мастер, и его реплики необходимо обновлять одновременно. Соединение мастера одной версии с репликой другой версии скорее всего приведет к потере данных.

Еще одна тонкость: реплику нельзя обновлять с помощью `pg_upgrade`. Такое обновление не дает 100% гарантии того, что два идентичных экземпляра после обновления также получатся идентичными.

Обычный способ состоит в том, чтобы забыть про существующие реплики и создать их заново. Для этого надо создать базовую резервную копию мастера и развернуть из нее новую реплику (процедура подробно рассматривается в курсе DBA3). Понятно, что это достаточно длительная процедура.

Если речь идет об операционной системе семейства Unix и при обновлении мастера использовался режим жестких ссылок, то время развертывания новой реплики можно существенно ускорить, получив примерно такой же выигрыш по времени, который дает `pg_upgrade` для мастера. Для этого надо правильным образом воспользоваться утилитой `rsync`.

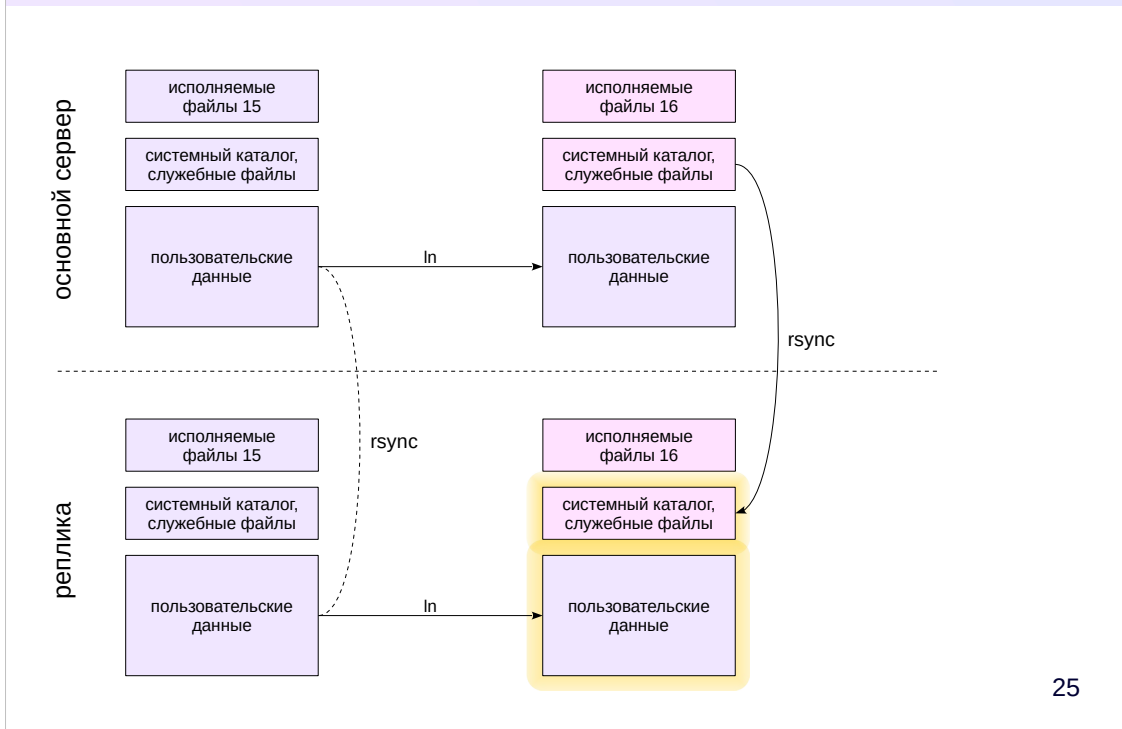
Физическая репликация



Исходная позиция: мастер обновлен, но новый сервер еще не запускался. Реплика выключена, причем перед выключением она полностью догнала мастер (таким образом файлы данных совпадают).

На реплике устанавливается новая версия сервера, но кластер пока не инициализируется.

Физическая репликация



Дальше запускается rsync так, чтобы в одной команде скопировать с мастера *оба кластера*, старый и новый, на реплику.

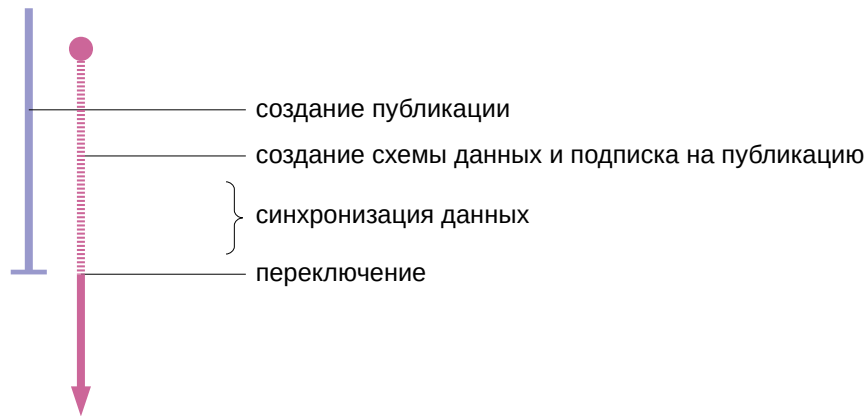
В таком режиме rsync понимает, что:

- старый и новый кластеры разделяют общие файлы данных;
- файлы данных старого кластера на мастере и на реплике совпадают (чтобы не зависеть от объема, сравниваются только размеры файлов, а не их содержимое);
- следовательно, файлы данных не надо физически копировать на реплику, а достаточно проставить жесткие ссылки.

И rsync остается скопировать только файлы, относящиеся к системному каталогу, а они имеют небольшой размер.

Этот процесс описан в документации к pg_upgrade (<https://postgrespro.ru/docs/postgresql/16/pgupgrade>), в том числе приведены и конкретные команды, которые необходимо выполнить. К сожалению, документация не объясняет, почему необходимы именно такие команды. Для того, чтобы разобраться в этом вопросе детально, полезно ознакомиться с перепиской разработчиков:

<https://www.postgresql.org/message-id/flat/54C95E66.1000309@agliodbs.com>



- + возможно обновление без или с минимальным прерыванием обслуживания
- сложная настройка, не реплицируются схема данных и команды DDL, требуется дополнительное место на диске и т. д.

Можно выполнить обновление основной версии без прерывания обслуживания (или с минимальным прерыванием), если использовать логическую репликацию.

Для этого требуется установить и настроить необходимым образом новый экземпляр. На старом сервере создаются публикации всех изменений во всех базах данных. На новом сервере создаются подписки на эти публикации с синхронизацией данных. После того как данные синхронизированы, остается переключить пользователей на новый сервер, удалить подписку и остановить старый сервер.

Штатная логическая репликация появилась в PostgreSQL 10.

Это решение активно развивается, но пока далеко от идеала: не реплицируются схема данных, команды DDL, данные последовательностей. Все это делает такой подход более сложным и менее привлекательным, чем использование `pg_upgrade`.

Обновление на дополнительный выпуск —
простая замена исполняемых файлов

физическая репликация может помочь не прерывать обслуживание

Обновление основной версии:

резервная копия (pg_dumpall)

утилита обновления (pg_upgrade)

логическая репликация

наличие физической репликации усложняет процедуру

сразу после — сделать физическую резервную копию

1. В кластере PostgreSQL 15 создайте пользователя с аутентификацией по паролю.
2. От имени нового пользователя создайте в базе данных таблицу со столбцом типа `uom`.
3. Обновите кластер 15 на версию 16 с помощью логической резервной копии. При обновлении сделайте так, чтобы все данные оказались размещены в отдельном табличном пространстве.
4. Проверьте корректность обновления: доступность сервера, аутентификацию пользователя, содержимое таблицы.

1. PostgreSQL 15 уже установлен, в каталоге `/var/postgresql/15/prod` инициализирован кластер, сервер работает на порту 5433. Локальные соединения безусловно разрешены, при подключении по TCP/IP настроена парольная аутентификация. В кластере создана единственная роль `postgres` с правами суперпользователя.

Для начального подключения к кластеру используйте команду
`psql -p 5433 -U postgres`

Рекомендуем остановить основной кластер `main`, чтобы при выполнении практики не подключиться к нему по ошибке.

При создании нового пользователя укажите пароль. Не забудьте изменить настройки аутентификации для нового пользователя в `pg_hba.conf`.

2. Тип данных `uom` доступен в одноименном расширении (см. тему «Расширения»).

3. PostgreSQL 16 уже установлен, в каталоге `/var/postgresql/16/prod` инициализирован кластер аналогично кластеру `prod` версии 15.

Чтобы разместить данные в отдельном табличном пространстве, предварительно создайте его и отредактируйте файл с резервной копией (добавьте табличное пространство по умолчанию для БД).

Чтобы сохранить аутентификацию, внесите в файл `pg_hba.conf` те же изменения, что были сделаны в старом кластере.

1. Пользователь и аутентификация в кластере 15

Остановим основной кластер, чтобы случайно к нему не подключиться, и запустим кластер 15 prod:

```
student$ sudo pg_ctlcluster 16 main stop
student$ sudo pg_ctlcluster 15 prod start
student$ pg_lsclusters
```

Ver	Cluster	Port	Status	Owner	Data directory	Log file
15	prod	5433	online	postgres	/var/lib/postgresql/15/prod	/var/log/postgresql/postgresql-15-prod.log
16	main	5432	down	postgres	/var/lib/postgresql/16/main	/var/log/postgresql/postgresql-16-main.log
16	prod	5433	down	postgres	/var/lib/postgresql/16/prod	/var/log/postgresql/postgresql-16-prod.log
16	slow	5432	down	postgres	/var/lib/postgresql/16/slow	/var/log/postgresql/postgresql-16-slow.log

```
student$ psql -p 5433 -U postgres
```

Пользователь dbuser:

```
| => CREATE USER dbuser PASSWORD 'mypassword';
| CREATE ROLE
```

Добавляем в начало pg_hba.conf правило для нового пользователя:

```
student$ sudo sed -i '1s/^/local all dbuser scram-sha-256\n/' /etc/postgresql/15/prod/pg_hba.conf
| => SELECT pg_reload_conf();
|
| pg_reload_conf
| -----
| t
| (1 row)
```

Вот что получилось:

```
student$ sudo egrep '^[^#]' /etc/postgresql/15/prod/pg_hba.conf
local all dbuser scram-sha-256
local all all trust
host all all 127.0.0.1/32 scram-sha-256
host all all ::1/128 scram-sha-256
local replication all trust
host replication all 127.0.0.1/32 scram-sha-256
host replication all ::1/128 scram-sha-256
```

Добавим пароль в .pgpass, чтобы не вводить его вручную (при выполнении задания этот шаг лучше пропустить: будет непонятно, срабатывает аутентификация по паролю или безусловный доступ, включенный по умолчанию).

```
student$ echo '*:*:*:dbuser:mypassword' > ~/.pgpass
student$ chmod 0600 ~/.pgpass
```

2. Таблица в кластере 15

Сначала создадим базу данных.

```
| => CREATE DATABASE admin_upgrade;
| CREATE DATABASE
| => \c admin_upgrade
| You are now connected to database "admin_upgrade" as user "postgres".
```

Тип данных uom доступен в расширении uom, установим версию 1.1.

```
student$ sudo make install -C /home/student/uom PG_CONFIG=/usr/lib/postgresql/15/bin/pg_config
make: Entering directory '/home/student/uom'
/bin/mkdir -p '/usr/share/postgresql/15/extension'
/bin/mkdir -p '/usr/share/postgresql/15/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/15/extension/'
/usr/bin/install -c -m 644 ./uom--1.0.sql ./uom--1.2.sql ./uom--1.0--1.1.sql
```

```
./uom--1.1--1.2.sql '/usr/share/postgresql/15/extension/'
make: Leaving directory '/home/student/uom'
```

```
| => CREATE EXTENSION uom VERSION '1.1';
| CREATE EXTENSION
```

Вспомним, что для создания таблицы пользователю понадобится право CREATE на схему. Суперпользователь должен предоставить ему это право:

```
| => GRANT CREATE ON SCHEMA public TO dbuser;
| GRANT
```

В сеансе пользователя dbuser создадим таблицу.

```
student$ psql -p 5433 -h localhost -U dbuser -d admin_upgrade
```

```
|| => CREATE TABLE test(id serial, length uom);
|| CREATE TABLE
|| => INSERT INTO test(length) VALUES ((500, 'км')::uom);
|| INSERT 0 1
|| => INSERT INTO test(length) VALUES ((8, 'мм')::uom);
|| INSERT 0 1
```

3. Логическая резервная копия и обновление на версию 16

```
student$ /usr/lib/postgresql/15/bin/pg_dumpall -U postgres -p 5433 > ~/dump.sql
```

Добавим табличное пространство по умолчанию:

```
student$ grep 'CREATE DATABASE admin_upgrade' ~/dump.sql | fold -sw 100
```

```
CREATE DATABASE admin_upgrade WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE_PROVIDER = libc
LOCALE = 'en_US.UTF-8';
```

```
student$ sed -i 's/\(CREATE DATABASE admin_upgrade WITH\)/\1 TABLESPACE = ts/' ~/dump.sql
```

```
student$ grep 'CREATE DATABASE admin_upgrade' ~/dump.sql | fold -sw 100
```

```
CREATE DATABASE admin_upgrade WITH TABLESPACE = ts TEMPLATE = template0 ENCODING = 'UTF8'
LOCALE_PROVIDER = libc LOCALE = 'en_US.UTF-8';
```

Останавливаем сервер.

```
student$ sudo pg_ctlcluster 15 prod stop
```

Настройки postgresql.conf не переносим, хотя в реальной жизни это, конечно, необходимо.

Изменяем pg_hba.conf так же, как для версии 15:

```
student$ sudo sed -i '1s/^/local all dbuser scram-sha-256\n/' /etc/postgresql/16/prod/pg_hba.conf
```

Создаем каталог для табличного пространства.

```
student$ sudo rm -rf /var/lib/postgresql/ts_dir
```

```
student$ sudo mkdir /var/lib/postgresql/ts_dir
```

```
student$ sudo chown postgres: /var/lib/postgresql/ts_dir
```

Стартуем кластер 16 и создаем табличное пространство от имени суперпользователя.

```
student$ sudo pg_ctlcluster 16 prod start
```

```
student$ psql -p 5433 -U postgres
```

```
| => CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir';
| CREATE TABLESPACE
```

Устанавливаем расширение:

```
student$ sudo make install -C /home/student/uom PG_CONFIG=/usr/lib/postgresql/16/bin/pg_config
```

```
make: Entering directory '/home/student/uom'
/bin/mkdir -p '/usr/share/postgresql/16/extension'
/bin/mkdir -p '/usr/share/postgresql/16/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/16/extension/'
```

```
/usr/bin/install -c -m 644 ./uom--1.0.sql ./uom--1.2.sql ./uom--1.0--1.1.sql
./uom--1.1--1.2.sql '/usr/share/postgresql/16/extension/'
make: Leaving directory '/home/student/uom'
```

Восстанавливаем кластер из резервной копии:

```
student$ sudo mv ~/dump.sql /var/lib/postgresql/
```

```
=> \i /var/lib/postgresql/dump.sql
```

```
SET
SET
SET
CREATE ROLE
ALTER ROLE
psql:/var/lib/postgresql/dump.sql:16: ERROR:  role "postgres" already exists
ALTER ROLE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
CREATE DATABASE
ALTER DATABASE
You are now connected to database "admin_upgrade" as user "postgres".
SET
SET
SET
SET
SET
set_config
-----
(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 2
setval
-----
      2
(1 row)

GRANT
You are now connected to database "postgres" as user "postgres".
```

```

SET
SET
SET
SET
SET
  set_config
-----

(1 row)

SET
SET
SET
SET

```

При создании роли postgres выдается ошибка, поскольку такая роль уже существует; это нормально.

4. Проверка работоспособности

```
student$ psql -p 5433 -h localhost -U dbuser -d admin_upgrade
```

```

=> \conninfo

You are connected to database "admin_upgrade" as user "dbuser" on host "localhost"
(address "127.0.0.1") at port "5433".
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)

=> SHOW server_version;

      server_version
-----
16.2 (Ubuntu 16.2-1ubuntu4)
(1 row)

=> \dx uom

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
uom    | 1.2     | public | Единицы измерения. uom--1.0--1.1.sql
(1 row)

=> SELECT * from test;

 id | length
----+-----
  1 | (500,км)
  2 | (8,мм)
(2 rows)

=> SELECT pg_relation_filepath('test');

      pg_relation_filepath
-----
pg_tblspc/16384/PG_16_202307071/16386/16417
(1 row)

```