

# Резервное копирование Базовая резервная копия



## **Авторские права**

© Postgres Professional, 2018–2022

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

## **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](mailto: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.

Понятие физической резервной копии

Холодное резервирование

Горячее резервирование

## Базовая физическая копия — копия файловой системы кластера

- + быстрее, чем логическое резервирование
- + восстанавливается статистика
- можно восстановиться только на совместимой системе  
и на той же самой основной версии PostgreSQL
- выборочная копия невозможна, копируется весь кластер

Физическое резервирование подразумевает копирование всех файлов, относящихся к кластеру БД, то есть создание полной двоичной копии.

Копирование файлов работает быстрее, чем выгрузка SQL-команд при логическом резервировании; запустить сервер из созданной физической копии — дело нескольких минут, в отличие от восстановления из логической копии. Кроме того, нет необходимости заново собирать статистику — она также восстанавливается из копии.

Но есть и минусы. Из физической резервной копии можно восстановить систему только на совместимой платформе (та же ОС, та же разрядность, тот же порядок байтов в представлении чисел и т. п.) и только на той же основной версии PostgreSQL. Кроме того, невозможно сделать физическую копию отдельных баз данных кластера, возможно копирование только всего кластера целиком.

## Выключенный сервер

если выполнена контрольная точка, то нужна только копия ФС  
копию можно развернуть на другом сервере, независимо от PGDATA

- + простота
- требуется прерывание обслуживания

## Снимок файловой системы

как при неаккуратном выключении: при старте потребуется  
восстановление, но в копию войдут все нужные файлы журнала  
данные, в т. ч. табличные пространства, должны войти в один снимок

- + не надо останавливать сервер
- файловые системы, поддерживающие снимки, работают медленней

Смысл холодного резервирования состоит в том, чтобы сделать копию файловой системы в тот момент, когда она содержит согласованные данные. Восстановление из такой копии происходит просто: файлы разворачиваются, запускается сервер — и он сразу же готов к работе.

К сожалению, единственный вариант сделать такую копию — аккуратно (с выполнением контрольной точки) остановить сервер. Минус понятен: необходима остановка сервера (которая к тому же может оказаться длительной при большом объеме данных).

Время простоя можно сократить за счет предварительного выполнения rsync (или аналогичного инструмента) при работающем сервере. Тогда после останова сервера rsync докопирует только изменения (которых, предположительно, будет не много). Но простоя все равно не избежать.

Другой вариант — сделать копию несогласованных данных. Такая ситуация может возникнуть при неаккуратном отключении сервера или при создании снимка файловой системы (если ФС имеет такую возможность, и если все необходимые файлы попадают в один снимок).

В этом случае восстановление происходит аналогично, но при старте серверу потребуется выполнить восстановление согласованности. Это обычная автоматическая процедура восстановления после сбоя. Она не представляет проблемы, так как необходимые файлы журнала гарантированно попадут в копию, но она потребует некоторого времени.

<https://postgrespro.ru/docs/postgresql/13/backup-file>

## Холодная файловая копия

Файлы остановленного кластера можно скопировать и запустить с ними второй сервер. При этом не важно, был ли сервер остановлен корректно.

Создадим базу данных и таблицу.

```
student$ sudo pg_ctlcluster 13 alpha start
```

```
student$ psql
```

```
α=> CREATE DATABASE backup_base;
```

```
CREATE DATABASE
```

```
α=> \c backup_base
```

You are now connected to database "backup\_base" as user "student".

```
α=> CREATE TABLE t(s text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO t VALUES ('Привет, мир!');
```

```
INSERT 0 1
```

Аварийно останавливаем сервер и копируем файлы на сервер beta:

```
student$ sudo head -n 1 /var/lib/postgresql/13/alpha/postmaster.pid
```

```
6241
```

```
student$ sudo kill -9 6241
```

```
student$ sudo pg_ctlcluster 13 beta status
```

```
Error: /var/lib/postgresql/13/beta is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/beta
```

```
student$ sudo cp -rp /var/lib/postgresql/13/alpha /var/lib/postgresql/13/beta
```

Сам резервный сервер уже предварительно собран и установлен.

Beta восстанавливает согласованность и запускается:

```
student$ sudo pg_ctlcluster 13 beta start
```

```
student$ sudo tail -n 5 /var/log/postgresql/postgresql-13-beta.log
```

```
2024-01-16 12:15:13.144 MSK [6609] LOG:  database system was not properly shut down; automatic recovery in progress
2024-01-16 12:15:13.160 MSK [6609] LOG:  redo starts at 0/3000F20
2024-01-16 12:15:13.163 MSK [6609] LOG:  invalid record length at 0/301BEF8: wanted 24, got 0
2024-01-16 12:15:13.163 MSK [6609] LOG:  redo done at 0/301BED0
2024-01-16 12:15:13.476 MSK [6608] LOG:  database system is ready to accept connections
```

```
student$ psql -p 5433 -d backup_base
```

```
| β=> SELECT * FROM t;
```

```
|          s
|-----
|  Привет, мир!
| (1 row)
```

```
student$ sudo pg_ctlcluster 13 beta stop
```

## Работающий сервер

не просто несогласованные, но и динамически изменяющиеся данные (проблема неатомарного чтения и записи)

для восстановления согласованности необходимы журнальные записи от последней контрольной точки за все время копирования (в процессе копирования сервер может удалить часть файлов)

+ не требуется прерывание обслуживания

– нужны специальные инструменты

Горячее резервирование выполняется на работающем сервере, поэтому в копию совершенно точно попадут несогласованные данные.

Более того. При резервном копировании данные читаются не через буферный кеш, а напрямую из файлов. Содержимое файлов на диске, очевидно, изменяется во время копирования, а файловая система обычно не гарантирует атомарность чтения/записи 8-килобайтной страницы PostgreSQL. Поэтому в резервную копию будут попадать «безнадежные» страницы, к которым даже нельзя применить журнальные записи. Это первая сложность.

Вторая сложность состоит в том, что копирование файлов данных может занимать достаточно много времени. Но сервер, после выполнения очередной контрольной точки, может удалить часть файлов журнала, которые уже не нужны ему для восстановления после сбоя, но нужны для резервной копии.

Если не принять специальных мер, сделанная резервная копия будет непригодна для восстановления. Поэтому для горячего резервирования требуются специальные инструменты. PostgreSQL предоставляет низкоуровневый интерфейс, используя который можно реализовать надежное копирование. Этот интерфейс использует и штатная утилита `pg_basebackup`, и другие сторонние средства резервного копирования.

<https://postgrespro.ru/docs/postgresql/13/continuous-archiving.html#BACKUP-BASE-BACKUP>

## Задачи

управление резервным копированием и репликацией  
в частности, получение потока журнальных записей

## Обслуживается процессом wal sender

*max\_wal\_senders*

*wal\_level* = replica или logical, но не minimal

## Подключение

роль с атрибутом REPLICATION или SUPERUSER

разрешение на подключение в pg\_hba.conf

Для упрощения задачи, сервер PostgreSQL предоставляет протокол репликации — специальный протокол для управления как собственно репликацией (рассматривается в одноименном модуле), так и резервным копированием. В частности, он позволяет получать поток журнальных записей, которые генерирует сервер.

На сервере подключение по протоколу репликации обслуживается процессом wal sender. Он похож на обычный обслуживающий процесс, который запускается при обычном подключении клиента, но понимает не SQL, а специальные команды. Число одновременно работающих процессов wal\_sender ограничено значением параметра *max\_wal\_senders*.

Уровень журнала должен быть не ниже, чем replica. Дело в том, что на уровне minimal такие команды, как CREATE TABLE AS SELECT, CREATE INDEX, COPY FROM, не попадают в журнал: их долговечность обеспечивается тем, что данные не остаются в оперативной памяти, а сразу записываются на диск. Этого достаточно для восстановления после сбоя и из холодной копии, но недостаточно для восстановления из горячей копии.

Чтобы использовать протокол репликации, клиент должен подключаться к серверу под ролью, имеющей атрибут REPLICATION (либо под суперпользователем). В pg\_hba.conf надо разрешить подключение этой роли к базе данных replication (это, конечно, не название БД, а ключевое слово). Причем разрешения для all недостаточно, replication должен быть разрешен отдельно.

<https://postgrespro.ru/docs/postgresql/13/protocol-replication>

## Серверный объект для получения журнальных записей

помнит, какая запись была передана клиенту последней  
число слотов ограничено *max\_replication\_slots*

Если используется слот, то сегмент WAL не удаляется  
контрольными точками, пока все его записи не переданы  
клиенту

позволяет клиенту получать данные в удобном режиме,  
в том числе отключаться на время  
при отключении журнальные файлы будут накапливаться на сервере  
мониторинг состояния: *pg\_replication\_slots*

Чтобы сервер не удалил необходимые файлы WAL преждевременно, можно применять слот репликации. Если поток журнальных записей идет через слот, то слот помнит, какие записи уже были переданы клиенту. Наличие слота не позволит серверу удалять файл WAL до тех пор, пока клиент не получит все записи из этого файла.

Использование слота позволяет клиенту не беспокоиться о том, что сервер сотрет файл журнала раньше времени. Клиент даже может отключиться и затем через какое-то время подключиться вновь и продолжить получать журнальные записи с того момента, на котором остановился.

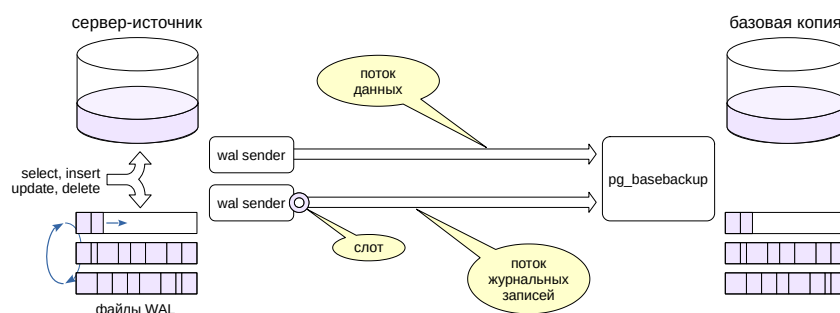
Но надо иметь в виду, что при отключении клиента файлы журнала будут накапливаться на сервере и могут занять все свободное место. Поэтому каждый созданный слот следует добавлять в мониторинг (представление *pg\_replication\_slots*) и своевременно удалять ненужные слоты.

Общее количество слотов, которые могут быть созданы, ограничено конфигурационным параметром *max\_replication\_slots*.

Более подробно применение слотов рассматривается в модуле «Репликация».

<https://postgrespro.ru/docs/postgresql/13/warm-standby.html#STREAMING-REPLICATION-SLOTS>





два потока,  $max\_wal\_senders \geq 2$

по умолчанию используется временный слот репликации

--slot=имя\_слота

--create-slot

--no-slot

Для выполнения копирования утилита pg\_basebackup использует два подключения по протоколу репликации: одно для передачи данных и одно — для передачи потока журнальных записей, которые генерирует работающий сервер во время копирования. Поэтому для pg\_basebackup значение параметра *max\_wal\_senders* должно быть не менее двух.

Для передачи журнальных записей pg\_basebackup, начиная с 10-й версии, PostgreSQL по умолчанию использует временный слот репликации, который существует только на время соединения и удаляется при завершении работы pg\_basebackup.

Однако в параметрах утилиты можно указать и имя обычного слота, который должен существовать на момент запуска утилиты или создаваться ей.

<https://postgrespro.ru/docs/postgresql/13/app-pgbasebackup>

## Немедленное развертывание нового экземпляра

`--format=plain`

удаленный запуск на сервере, где будет развернут экземпляр копирует файлы и каталоги кластера (PGDATA) в указанный каталог копирует табличные пространства по тем же абсолютным путям, но можно сопоставить и другие пути (`--tablespace-mapping`)

## Резервная копия для последующего использования

`--format=tar`

`--gzip` или `--compress=0..9` для сжатия

удаленный или локальный запуск

помещает PGDATA в `base.tar`, файлы журнала в `pg_wal.tar`

помещает каждое табличное пространство в отдельный файл `OID.tar`, пути могут быть изменены в файле `tablespace_map`

10

Если предполагается немедленно развернуть новый сервер из резервной копии, удобно вызывать `pg_basebackup` с форматом `plain` (используется по умолчанию), запуская его на целевом сервере. Утилита удаленно подключается к серверу-источнику и создает локальные каталоги и файлы, соответствующие каталогам и файлам основного сервера. Таким образом, новый сервер можно запускать, как только отработает `pg_basebackup`.

Табличные пространства будут скопированы по тем же абсолютным путям, что и на сервере-источнике (поэтому в таком режиме `pg_basebackup` нельзя запускать на сервере-источнике). Однако при необходимости можно переназначить пути для табличных пространств, указав соответствие в параметрах утилиты.

Если же копия выполняется в рамках обычной политики резервного копирования, удобно воспользоваться форматом `tar`. В этом случае `pg_basebackup` можно запускать как на сервере-источнике, так и удаленно. Основной каталог кластера PGDATA будет сохранен в файле `base.tar`, журналы — в файле `pg_wal.tar`, а табличные пространства — каждое в своем собственном `tar`-файле, имя которого будет совпадать с OID табличного пространства. Файлы могут быть сжаты, если указать соответствующие ключи утилиты.

Для восстановления из такой копии сначала потребуется развернуть `tar`-файлы по правильным путям. При этом табличные пространства можно разместить по новым путям, но потребуется отредактировать файл `tablespace_map` перед запуском сервера.

## Базовая резервная копия

Теперь мы хотим сделать базовую копию работающего сервера.

```
student$ sudo pg_ctlcluster 13 alpha start
```

Значения параметров по умолчанию позволяют сразу использовать протокол репликации:

```
student$ psql -U postgres
```

```
α=> SELECT name, setting
FROM pg_settings
WHERE name IN ('wal_level', 'max_wal_senders', 'max_replication_slots');
```

name	setting
max_replication_slots	10
max_wal_senders	10
wal_level	replica

(3 rows)

Разрешение на локальное подключение по протоколу репликации в pg\_hba.conf также прописано по умолчанию (хотя это и зависит от конкретной пакетной сборки):

```
α=> SELECT type, database, user_name, address, auth_method
FROM pg_hba_file_rules()
WHERE 'replication' = ANY(database);
```

type	database	user_name	address	auth_method
local	{replication}	{all}		trust
host	{replication}	{all}	127.0.0.1	md5
host	{replication}	{all}	:::1	md5

(3 rows)

Чтобы утилита pg\_basebackup могла подключиться к серверу под ролью student, эта роль должна иметь атрибут REPLICATION:

```
α=> \du student
```

Role name	List of roles	Member of
	Attributes	
student	Create role, Create DB, Replication	{pg_read_all_stats}

Выполним команду pg\_basebackup. В нашем случае и сервер-источник, и резервная копия будут располагаться на одном сервере.

Если бы мы использовали табличные пространства, дополнительно пришлось бы указать для них другие пути в ключе --tablespace-mapping, но в данном случае этого не требуется.

Для мониторинга добавим ключ --progress. Ту же информацию можно получить в реальном времени из представления pg\_stat\_progress\_basebackup.

```
student$ pg_basebackup --pgdata=/home/student/backup --progress
```

```
waiting for checkpoint
 0/40184 kB (0%), 0/1 tablespace
40194/40194 kB (100%), 0/1 tablespace
40194/40194 kB (100%), 1/1 tablespace
```

По умолчанию в начале копирования выполняется «протяженная» контрольная точка в соответствии с обычной настройкой. Это может занять заметное время: если контрольные точки выполняются по расписанию, то соответствующую долю от значения параметра checkpoint\_timeout.

```
α=> SHOW checkpoint_timeout; SHOW checkpoint_completion_target;
```

```
checkpoint_timeout
-----
5min
(1 row)

checkpoint_completion_target
-----
0.5
(1 row)
```

Если требуется выполнить контрольную точку как можно быстрее, надо указать ключ `--checkpoint=fast`.

Проверим содержимое каталога с данными, в который была записана базовая копия:

```
student$ ls -l /home/student/backup

total 300
-rw----- 1 student student    224 янв 16 12:15 backup_label
-rw----- 1 student student    224 янв 16 12:15 backup_label.old
-rw----- 1 student student 219599 янв 16 12:15 backup_manifest
drwx----- 7 student student  4096 янв 16 12:15 base
drwx----- 2 student student  4096 янв 16 12:15 global
drwx----- 2 student student  4096 янв 16 12:15 pg_commit_ts
drwx----- 2 student student  4096 янв 16 12:15 pg_dynshmem
drwx----- 4 student student  4096 янв 16 12:15 pg_logical
drwx----- 4 student student  4096 янв 16 12:15 pg_multixact
drwx----- 2 student student  4096 янв 16 12:15 pg_notify
drwx----- 2 student student  4096 янв 16 12:15 pg_replslot
drwx----- 2 student student  4096 янв 16 12:15 pg_serial
drwx----- 2 student student  4096 янв 16 12:15 pg_snapshots
drwx----- 2 student student  4096 янв 16 12:15 pg_stat
drwx----- 2 student student  4096 янв 16 12:15 pg_stat_tmp
drwx----- 2 student student  4096 янв 16 12:15 pg_subtrans
drwx----- 2 student student  4096 янв 16 12:15 pg_tblspc
drwx----- 2 student student  4096 янв 16 12:15 pg_twophase
-rw----- 1 student student     3 янв 16 12:15 PG_VERSION
drwx----- 3 student student  4096 янв 16 12:15 pg_wal
drwx----- 2 student student  4096 янв 16 12:15 pg_xact
-rw----- 1 student student    88 янв 16 12:15 postgresql.auto.conf
```

Все необходимые файлы журнала находятся в каталоге `pg_wal`:

```
student$ ls -l /home/student/backup/pg_wal/

total 16388
-rw----- 1 student student 16777216 янв 16 12:15 00000001000000000000000004
drwx----- 2 student student   4096 янв 16 12:15 archive_status
```

## Восстановление из базовой резервной копии

Скопируем базовую копию в каталог данных сервера `beta`.

```
student$ sudo pg_ctlcluster 13 beta status

pg_ctl: no server running

student$ sudo rm -rf /var/lib/postgresql/13/beta/*

student$ sudo cp -r /home/student/backup/* /var/lib/postgresql/13/beta

student$ sudo chown -R postgres /var/lib/postgresql/13/beta
```

Запускаем второй сервер.

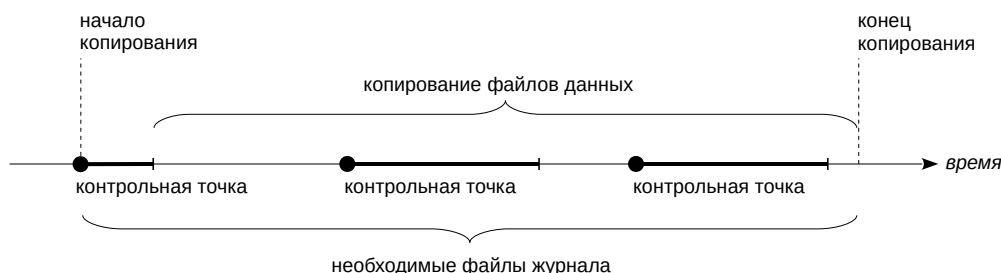
```
student$ sudo pg_ctlcluster 13 beta start
```

Теперь оба сервера работают одновременно и независимо. Проверим:

```
student$ psql -p 5433 -d backup_base
```

```
β=> SELECT * FROM t;

      s
-----
Привет, мир!
(1 row)
```



на время копирования устанавливается *full\_page\_writes = on*  
в самом начале выполняется контрольная точка (быстро или обычно)

для борьбы  
с неатомарностью  
чтения и записи

Общий алгоритм изготовления резервной копии на низком уровне одинаков как для *pg\_basebackup*, так и для любых сторонних средств (которые могут потребоваться, поскольку *pg\_basebackup* предоставляет только самую базовую функциональность).

1. Надо сообщить серверу о том, что начинается резервное копирование.

При этом, во-первых, на время копирования устанавливается параметр *full\_page\_writes*: при первом изменении страницы после контрольной точки полный образ этой страницы записывается в журнал.

При восстановлении журнальные записи будут применяться не к страницам в файле (которые, как мы видели, могут быть прочитаны в рассогласованном состоянии), а к образу страницы из журнала.

Во-вторых, выполняется контрольная точка. Предусмотрено два режима: быстрое выполнение (что может привести к пиковой нагрузке на дисковую подсистему) и протяженное (которое определяется обычным параметром *checkpoint\_completion\_target*).

2. После прохождения контрольной точки можно копировать файлы данных любым удобным способом.

3. После того, как все скопировано, надо сообщить серверу, что резервное копирование завершено.

4. Кроме того, так или иначе надо обеспечить попадание в резервную копию всех журнальных записей, сгенерированных с начала копирования и до его окончания.

## Начало копирования: `pg_start_backup`

монопольный и немонопольный режимы  
выполняет контрольную точку, устанавливает *full\_page\_writes*  
возвращает начальную позицию в журнале

## Конец копирования: `pg_stop_backup`

возвращает конечную позицию в журнале

## Копирование файлов

либо репликационный протокол, либо любые средства ОС

## Копирование журнала от начальной до конечной позиции

либо репликационный протокол, либо средства ОС + *wal\_keep\_size*

13

Описанный алгоритм резервного копирования можно использовать, чтобы реализовать свою собственную утилиту копирования.

Начало резервного копирования выполняется функцией `pg_start_backup`. Она устанавливает параметр *full\_page\_writes* и выполняет контрольную точку, как рассматривалось выше. Возвращаемое этой функцией значение — позиция в журнале, начиная с которой в копию должны попасть журнальные записи.

Начать резервное копирования можно в монопольном режиме (он чуть проще, но считается устаревшим) либо в немонопольном (позволяет делать несколько резервных копий одновременно).

Конец копирования отмечается функцией `pg_stop_backup`. Она возвращает конечную позицию в журнале.

Все журнальные записи между начальной и конечной позициями надо поместить в резервную копию. Поскольку при своей работе сервер может удалять журнальные файлы, которые не требуются для восстановления, надо либо пользоваться протоколом репликации и слотом, получая записи во время копирования (как `pg_basebackup`), либо (что менее удобно) копировать файлы средствами ОС, установив параметр *wal\_keep\_size* (до версии 13 нужно указывать *wal\_keep\_segments*, при этом  $wal\_keep\_size = wal\_keep\_segments * wal\_segment\_size$ ).

Копирование файлов данных также можно выполнить через протокол репликации (как `pg_basebackup`), так и средствами ОС. Можно копировать файлы в несколько потоков, использовать `rsync` и т. п.

<https://postgrespro.ru/docs/postgresql/13/continuous-archiving.html#BACKUP-LOWLEVEL-BASE-BACKUP>

## Манифест

файл `backup_manifest`  
генерируется по умолчанию

## Проверка копии

утилита `pg_verifybackup`  
проверяет файлы по списку из манифеста

## Проверка файлов данных

утилита `pg_basebackup`  
по умолчанию проверяет контрольные суммы

утилита `pg_checksums`  
проверяет контрольные суммы файлов отношений

При инициализации кластера можно включить расчет и сохранение контрольных сумм страниц (`initdb -k`). Утилита **`pg_checksums`** позволяет в дальнейшем включать и выключать этот режим, но требует остановки сервера.

<https://postgrespro.ru/docs/postgresql/13/app-pgchecksums>

Контрольные суммы вычисляются при каждом изменении страницы и сохраняются в ее заголовке. Сравнение вычисленной и сохраненной контрольных сумм производится при чтении страницы в буферный кеш, утилитой **`pg_basebackup`** при формировании копии и утилитой **`pg_checksums`** в режиме проверки.

Утилита **`pg_basebackup`** включает в копию файл **`backup_manifest`**, содержащий информацию о файлах (имена, размеры, вычисленные контрольные суммы) и журнальных записях (начальная и конечная позиции WAL, линия времени) и контрольную сумму манифеста.

<https://postgrespro.ru/docs/postgresql/13/backup-manifest-format>

Утилита **`pg_verifybackup`**, основываясь на информации из манифеста, проверяет наличие файлов, соответствие контрольных сумм, возможность чтения и разбора записей WAL, необходимых для восстановления. Нужно понимать, что успешное выполнение всех проверок не гарантирует отсутствия ошибок в резервной копии, критерием ее работоспособности может быть только возможность безошибочного восстановления и бесперебойной работы сервера.

<https://postgrespro.ru/docs/postgresql/13/app-pgverifybackup>

## Проверка целостности

Резервная копия содержит файл манифеста, вот его начало и конец:

```
student$ head -n 5 ~/backup/backup_manifest
```

```
{ "PostgreSQL-Backup-Manifest-Version": 1,
  "Files": [
    { "Path": "backup_label", "Size": 224, "Last-Modified": "2024-01-16 09:15:28 GMT", "Checksum-Algorithm": "CRC32C", "Checksum": "14719530" },
    { "Path": "pg_xact/0000", "Size": 8192, "Last-Modified": "2024-01-16 09:15:27 GMT", "Checksum-Algorithm": "CRC32C", "Checksum": "376e144c" },
    { "Path": "global/2677", "Size": 16384, "Last-Modified": "2022-07-14 08:23:57 GMT", "Checksum-Algorithm": "CRC32C", "Checksum": "7d22c055" },
```

```
student$ tail -n 6 ~/backup/backup_manifest
```

```
{ "Path": "global/pg_control", "Size": 8192, "Last-Modified": "2024-01-16 09:15:28 GMT", "Checksum-Algorithm": "CRC32C", "Checksum": "43872087" }
],
"WAL-Ranges": [
  { "Timeline": 1, "Start-LSN": "0/4000028", "End-LSN": "0/4000100" }
],
"Manifest-Checksum": "8576ce3318b828ab11e801515df2a80e09877498c890c530882dcff3f48567e7"}
```

Проверим целостность копии утилитой pg\_verifybackup:

```
student$ /usr/lib/postgresql/13/bin/pg_verifybackup ~/backup
```

```
backup successfully verified
```

Утилита проверяет по манифесту наличие файлов, их контрольные суммы, а также наличие и возможность чтения всех записей WAL, необходимых для восстановления.



Базовая резервная копия —  
полная копия кластера, включая табличные пространства,  
плюс журнал для восстановления согласованности

Горячее резервирование позволяет не останавливать сервер

## Инструменты

- протокол репликации
- слот репликации
- низкоуровневый API резервирования

1. В первом кластере создайте табличное пространство и базу данных с таблицей в этом пространстве.
2. Сделайте базовую резервную копию кластера с помощью `pg_basebackup` в формате `tar` со сжатием.
3. Разверните второй кластер из этой резервной копии. Табличное пространство разместите в другом каталоге, изменив файл `tablespace_map`.
4. Запустите второй сервер и проверьте его работоспособность.
5. Удалите базу данных и табличное пространство в обоих кластерах.

## 1. Табличное пространство и база данных

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

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

Табличные пространства может создавать только суперпользователь:

```
student$ psql -U postgres -c "CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir'"
```

```
CREATE TABLESPACE
```

```
student$ psql -U postgres -c "ALTER TABLESPACE ts OWNER TO student"
```

```
ALTER TABLESPACE
```

```
α=> CREATE DATABASE backup_base;
```

```
CREATE DATABASE
```

```
α=> \c backup_base
```

You are now connected to database "backup\_base" as user "student".

```
α=> CREATE TABLE t(s text) TABLESPACE ts;
```

```
CREATE TABLE
```

```
α=> INSERT INTO t VALUES ('Привет, мир!');
```

```
INSERT 0 1
```

## 2. Базовая резервная копия

Запускаем pg\_basebackup от имени роли student, указывая формат tar со сжатием:

```
student$ pg_basebackup --format=tar --gzip --pgdata=/home/student/backup
```

```
student$ ls -l /home/student/backup
```

```
total 5208
-rw----- 1 student student    344 янв 16 12:22 24576.tar.gz
-rw----- 1 student student 219822 янв 16 12:22 backup_manifest
-rw----- 1 student student 5086870 янв 16 12:22 base.tar.gz
-rw----- 1 student student   17075 янв 16 12:22 pg_wal.tar.gz
```

## 3. Восстановление из базовой резервной копии

Основной каталог данных кластера beta

```
student$ sudo pg_ctlcluster 13 beta status
```

```
Error: /var/lib/postgresql/13/beta is not accessible or does not exist
```

```
student$ sudo rm -rf /var/lib/postgresql/13/beta
```

```
student$ sudo mkdir /var/lib/postgresql/13/beta
```

Каталог для табличного пространства ts на сервере, который будет развернут из резервной копии:

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

Разворачиваем резервную копию:

```
student$ sudo tar -zxf /home/student/backup/base.tar.gz -C /var/lib/postgresql/13/beta
```

```
student$ sudo tar -zxf /home/student/backup/24576.tar.gz -C /var/lib/postgresql/ts_beta_dir
```

```
student$ sudo tar -zxf /home/student/backup/pg_wal.tar.gz -C /var/lib/postgresql/13/beta/pg_wal
```

Меняем владельца и разрешения:

```
student$ sudo chown -R postgres /var/lib/postgresql/13/beta
```

```
student$ sudo chmod -R 700 /var/lib/postgresql/13/beta
```

```
student$ sudo chown -R postgres /var/lib/postgresql/ts_beta_dir
```

В каталоге pg\_tblspc сейчас пусто:

```
student$ sudo ls /var/lib/postgresql/13/beta/pg_tblspc/
```

Символическая ссылка появится при старте сервера в соответствии с файлом tablespace\_map (который находился внутри base.tar):

```
student$ sudo cat /var/lib/postgresql/13/beta/tablespace_map
```

```
24576 /var/lib/postgresql/ts_dir
```

Изменяем в этом файле путь для табличного пространства:

```
student$ sudo sed -i 's/ts_dir/ts_beta_dir/' /var/lib/postgresql/13/beta/tablespace_map
```

## 4. Запуск и проверка

Запускаем сервер.

```
student$ sudo pg_ctlcluster 13 beta start
```

```
student$ sudo ls -l /var/lib/postgresql/13/beta/pg_tblspc/
```

```
total 0
```

```
lrwxrwxrwx 1 postgres root 31 янв 16 12:22 24576 -> /var/lib/postgresql/ts_beta_dir
```

```
student$ psql -p 5433 -d backup_base
```

```
| β=> \c backup_base
|
| You are now connected to database "backup_base" as user "student".
|
| β=> SELECT * FROM t;
|
|      s
| -----
| Привет, мир!
| (1 row)
```

## 5. Удаление базы данных и табличного пространства

Чтобы удалить БД, надо от нее отключиться.

```
α=> \c student
```

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

```
α=> DROP DATABASE backup_base;
```

```
DROP DATABASE
```

```
α=> DROP TABLESPACE ts;
```

```
DROP TABLESPACE
```

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

И для второго сервера:

```
| β=> \c student
|
| You are now connected to database "student" as user "student".
|
| β=> DROP DATABASE backup_base;
|
| DROP DATABASE
|
| β=> DROP TABLESPACE ts;
|
| DROP TABLESPACE
```

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