

Репликация Переключение на реплику



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

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.

Переключение на реплику

Особенности, связанные с файловым архивом

Возвращение в строй бывшего мастера

Причины

плановое переключение (switchover):
останов основного сервера для проведения технических работ
аварийное переключение (failover):
переход на реплику из-за сбоя основного сервера

Процедура

убедиться, что мастер остановлен
`$ pg_ctl promote` или
`promote_trigger_file` или
`pg_promote()`
автоматическое переключение: отсутствует
(для автоматизации требуется стороннее кластерное ПО)

Причиной перехода на резервный сервер может быть необходимость проведения технических работ на основном сервере — тогда переход выполняется в удобное время в штатном режиме (switchover). А может быть сбой основного сервера, и в таком случае переходить на резервный сервер нужно как можно быстрее, чтобы сократить время простоя системы (failover).

В любом случае сначала нужно убедиться, что мастер остановлен. Это очень важно, иначе данные на разных серверах «разойдутся», и свести их потом воедино — нетривиальная и неавтоматизируемая задача. Скорее всего, часть данных придется просто потерять.

Затем выполняется переход на реплику в ручном режиме. Автоматизация этого процесса возможна, но требует стороннего кластерного программного обеспечения (это обсуждается в модуле «Кластерные технологии»).

Переключение состоит в разрыве цикла восстановления. Для этого реплике посылается команда promote: либо командой `pg_ctl promote`, либо вызовом функции `pg_promote` из SQL. Другой вариант — задать имя файла в параметре `promote_trigger_file`. При появлении в системе файла с таким именем восстановление прерывается.

Еще один вариант: удалить файл `standby.signal` и перезапустить резервный сервер. Это не вполне «честный» способ, поскольку в этом случае реплика не поймет, что восстановление завершено, и не перейдет на новую линию времени. Дальше мы будем рассматривать только два первых варианта.

Применяются журнальные записи

уже полученные wal receiver, но еще не примененные startup

Завершаются процессы

wal receiver

startup

Запускаются процессы

wal writer

autovacuum launcher

archiver (зависит от настройки)

Переход на новую линию времени

Получив сигнал, сервер завершает восстановление и переходит в обычный режим работы.

Для этого он применяет уже полученные журнальные записи, которые еще не были применены.

Процессы wal receiver и startup завершают свою работу — на основном сервере они не нужны. А вот процессы wal writer и autovacuum launcher, наоборот, запускаются.

Кроме того, сервер переходит на новую линию времени.

С точностью до некоторых деталей, все происходит так же, как при окончании восстановления из резервной копии.

Настройка потоковой репликации

Настроим реплику так же, как делали в предыдущей теме, а потом перейдем на нее.

Создаем автономную резервную копию, попросив утилиту создать слот и необходимые файлы (postgresql.auto.conf с настройками и standby.signal).

```
student$ pg_basebackup --pgdata=/home/student/backup -R --slot=replica --create-slot
```

Выкладываем копию в каталог PGDATA сервера beta:

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/beta
```

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

Запускаем реплику:

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

Проверим настроенную репликацию. Выполним несколько команд на мастере:

```
α=> CREATE DATABASE replica_switchover;
```

CREATE DATABASE

```
α=> \c replica_switchover
```

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

```
α=> CREATE TABLE test(s text);
```

CREATE TABLE

```
α=> INSERT INTO test VALUES ('Привет, мир!');
```

INSERT 0 1

Проверим реплику:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_switchover
```

```
| β=> SELECT * FROM test;
```

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

Переход на реплику

Сейчас сервер beta является репликой (находится в режиме восстановления):

```
| β=> SELECT pg_is_in_recovery();
```

```
| pg_is_in_recovery |
|-----|
| t                |
| (1 row)           |
```

Повышаем реплику. В версии 13 появилась функция pg_promote(), которая выполняет то же действие.

```
student$ sudo pg_ctlcluster 13 beta promote
```

Теперь бывшая реплика стала полноценным экземпляром.

```
| β=> SELECT pg_is_in_recovery();
```

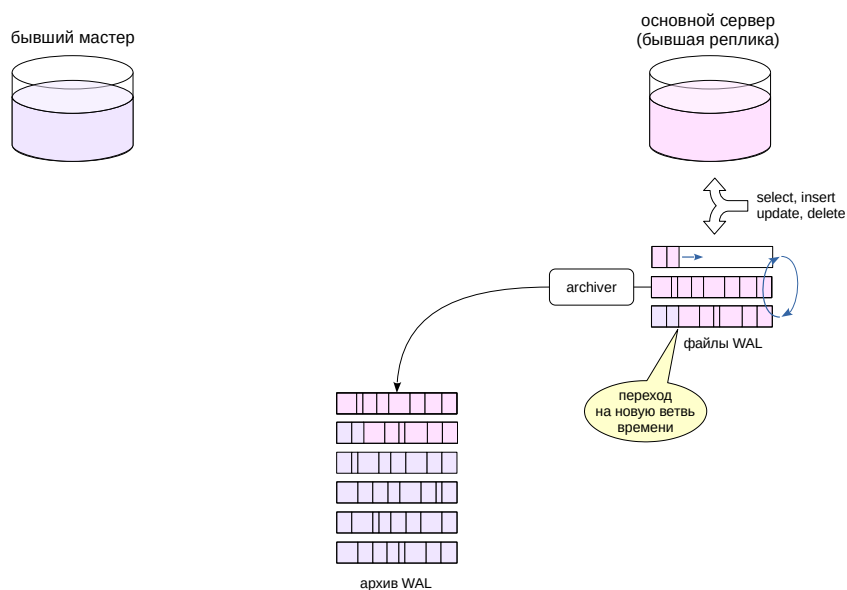
```
| pg_is_in_recovery |
|-----|
| f                |
| (1 row)           |
```

Мы можем изменять данные:

```
| β=> INSERT INTO test VALUES ('Я - бывшая реплика (новый мастер).');
```

```
| INSERT 0 1
```

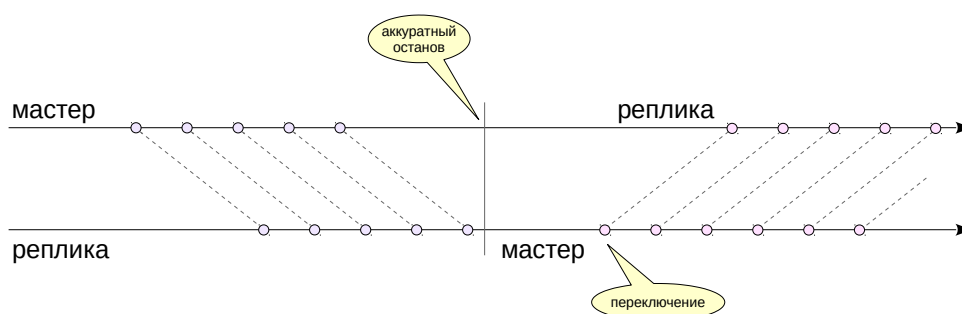

Возвращение в строй



Если переход на реплику не был вызван выходом сервера из строя, то нужен способ быстро вернуть старый мастер в строй — теперь уже в качестве реплики (failback).

Бывший мастер подключается к новому как реплика

допустимо, только если перед переключением реплика получила от мастера все журнальные записи



7

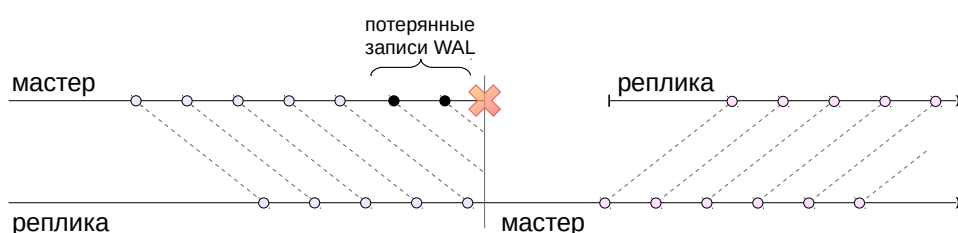
В случае аккуратной остановки мастера (останов в режимах `fast` или `smart`) все журнальные записи мастера скорее всего дойдут до реплики, хотя это и не гарантируется. Процесс останова организован таким образом, что сначала отключаются все обслуживающие процессы, затем выполняется контрольная точка, и только в самую последнюю очередь останавливается процесс `wal sender`, чтобы реплика успела получить запись WAL о контрольной точке.

Позицию в журнале мастера можно проверить с помощью утилиты `pg_controldata` («Latest checkpoint location»), а позицию на реплике покажет функция `pg_last_wal_receive_lsn()`. Поскольку функция `pg_last_wal_receive_lsn()` показывает *следующую* позицию, то ее значение должно *опережать* `latest_checkpoint_location` на длину записи (120 байт в PostgreSQL 13). Если это так, то бывший мастер можно непосредственно подключить к новому, изменив соответствующим образом конфигурационные параметры.

В случае останова мастера без выполнения контрольной точки (сбой или режим `immediate`), такое подключение в принципе невозможно: сервер не стартует, а в журнале сообщений будет зафиксирована ошибка.

Бывший мастер восстанавливается из резервной копии

абсолютно новая реплика, процесс занимает много времени
можно ускорить rsync, если с момента сбоя прошло немного времени
(но все равно долго для больших баз данных)



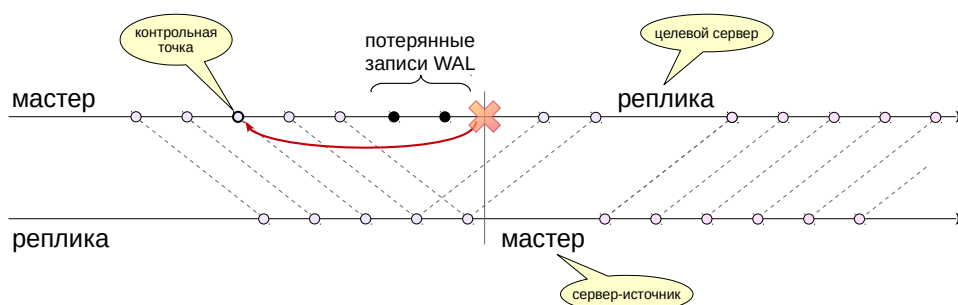
Если мастер был остановлен аварийно, велика вероятность того, что часть журнальных записей не успела дойти до реплики. В этом случае просто так подключать мастер нельзя.

Простой и надежный вариант — создать абсолютно новую реплику путем изготовления и развертывания базовой резервной копии. Однако для больших баз данных этот процесс может занимать много времени.

Вариант такого подхода — не использовать утилиту `pg_basebackup`, а сделать копию с помощью API резервирования с использованием утилиты `rsync`. Если выполнять копирование сразу после перехода на реплику, то большая часть файлов не должна успеть поменяться и процесс может пройти существенно быстрее. Но это, конечно, усложняет процесс.

Подготовка к восстановлению

«откатывает» потерянные записи WAL, заменяя соответствующие страницы целевого сервера страницами с сервера-источника копирует с сервера-источника все служебные файлы



9

Еще более быстрый вариант состоит в использовании штатной утилиты `pg_rewind`.

Утилита определяет место расхождения между двумя серверами, определяет ближайшую к нему общую контрольную точку, и, просматривая журнал, определяет все страницы, измененные с момента этой контрольной точки.

Найденные страницы (которых должно быть немного) заменяются страницами с сервера-источника (нового мастера). Кроме того, утилита копирует с сервера-источника все служебные файлы.

Дальше применяются все необходимые записи WAL с нового мастера. Фактически, это выполняет уже не утилита, а обычный процесс восстановления после запуска сервера. Чтобы восстановление началось с нужного момента, утилита создает управляющий файл `backup_label`.

Целевой сервер

все необходимые журнальные файлы должны сохраниться в `pg_wal` или в архиве (`--restore-target-wal`)

должны быть включены контрольные суммы или `wal_log_hints = on`

Сервер-источник

должен быть включен параметр `full_page_writes = on`

Утилита имеет ряд особенностей, ограничивающих ее применение. Необходимо, в числе прочего:

- Все сегменты WAL от текущего момента до найденной контрольной точки должны находиться в каталоге `pg_wal` целевого сервера или быть доступны для получения из архива, в этом случае надо задать параметр `restore_command` и ключ `--restore-target-wal`.
- Первое изменение данных после контрольной точки должно вызывать запись в WAL полной страницы. Параметра `full_page_writes = on` недостаточно, поскольку он не учитывает «незначительные» изменения страниц (hint bits). Дополнительно требуется, чтобы либо кластер был инициализирован с контрольными суммами страниц, либо нужно устанавливать параметр `wal_log_hints = on`.
- Целевой сервер должен быть остановлен аккуратно, с выполнением контрольной точки. Если это не так, утилита по умолчанию запустит целевой сервер и тут же остановит его корректно.
- На сервере-источнике заранее должен быть установлен параметр `full_page_writes = on` — причина та же, что и при восстановлении из резервной копии: утилита может скопировать страницы в рассогласованном состоянии.

Утилита pg_rewind

Между тем сервер alpha еще не выключен и тоже может изменять данные:

```
α=> INSERT INTO test VALUES ('Die hard');

INSERT 0 1
```

В реальности такой ситуации необходимо всячески избегать, поскольку теперь непонятно, какому серверу верить. Придется либо полностью потерять изменения на одном из серверов, либо придумывать, как объединить данные.

Наш выбор — потерять изменения, сделанные на первом сервере.

Мы планируем использовать утилиту pg_rewind, поэтому убедимся, что включены контрольные суммы на страницах данных:

```
α=> SHOW data_checksums;

data_checksums
-----
on
(1 row)
```

Этот параметр служит только для информации; изменить его нельзя — подсчет контрольных сумм задается при инициализации кластера или утилитой pg_checksums на остановленном сервере.

Остановим целевой сервер (alpha) некорректно.

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

14064

student$ sudo kill -9 14064
```

Создадим на сервере-источнике (beta) слот для будущей реплики:

```
| β=> SELECT pg_create_physical_replication_slot('replica');

pg_create_physical_replication_slot
-----
(replica,)
(1 row)
```

И проверим, что параметр full_page_writes включен:

```
| β=> SHOW full_page_writes;

full_page_writes
-----
on
(1 row)
```

Если целевой сервер не был остановлен корректно, утилита сначала запустит его в монопольном режиме и остановит с выполнением контрольной точки. Для запуска требуется наличие файла postgresql.conf в PGDATA.

```
postgres$ touch /var/lib/postgresql/13/alpha/postgresql.conf
```

В ключах утилиты pg_rewind надо указать каталог PGDATA целевого сервера и способ обращения к серверу-источнику: либо подключение от имени суперпользователя (если сервер работает), либо местоположение его каталога PGDATA (если он выключен).

```
postgres$ /usr/lib/postgresql/13/bin/pg_rewind -D /var/lib/postgresql/13/alpha --source-server='user=postgres port=5433' -R -P
```

```
pg_rewind: connected to server
pg_rewind: executing "/usr/lib/postgresql/13/bin/postgres" for target server to complete crash recovery
2022-05-20 11:08:24.483 GMT [14690] LOG:  database system was interrupted; last known up at 2022-05-20 11:08:15 GMT
2022-05-20 11:08:24.484 GMT [14690] LOG:  database system was not properly shut down; automatic recovery in progress
2022-05-20 11:08:24.487 GMT [14690] LOG:  redo starts at 0/5000ED0
2022-05-20 11:08:24.488 GMT [14690] LOG:  invalid record length at 0/501BF28: wanted 24, got 0
2022-05-20 11:08:24.488 GMT [14690] LOG:  redo done at 0/501BF00
```

```
PostgreSQL stand-alone backend 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)
backend- pg_rewind: servers diverged at WAL location 0/501BEC0 on timeline 1
pg_rewind: rewinding from last common checkpoint at 0/5000F08 on timeline 1
pg_rewind: reading source file list
pg_rewind: reading target file list
pg_rewind: reading WAL in target
pg_rewind: need to copy 54 MB (total source directory size is 86 MB)
    0/55371 kB (0%) copied
    448/55371 kB (0%) copied
    55371/55371 kB (100%) copied
pg_rewind: creating backup label and updating control file
pg_rewind: syncing target data directory
pg_rewind: Done!
```

В результате работы pg_rewind «откатывает» файлы данных на ближайшую контрольную точку до того момента, как пути серверов разошлись, а также создает файл backup_label, который обеспечивает применение нужных журналов для завершения восстановления.

Заглянем в backup_label:

```
student$ sudo cat /var/lib/postgresql/13/alpha/backup_label

START WAL LOCATION: 0/5000ED0 (file 00000001000000000000000005)
CHECKPOINT LOCATION: 0/5000F08
BACKUP METHOD: pg_rewind
BACKUP FROM: standby
START TIME: 2022-05-20 14:08:26 MSK
```

Ключом -R мы попросили утилиту создать сигнальный файл standby.signal и задать в конфигурационном файле строку соединения и имя слота.

```
student$ sudo cat /var/lib/postgresql/13/alpha/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=student passfile='/home/student/.pgpass' channel_binding=prefer host='/var/run/postgresql' port=5432 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
primary_conninfo = 'user=postgres passfile='/var/lib/postgresql/.pgpass' channel_binding=prefer port=5433 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_protocol_version=TL
student$ sudo ls -l /var/lib/postgresql/13/alpha/standby.signal

-rw----- 1 postgres postgres 0 мая 20 14:08 /var/lib/postgresql/13/alpha/standby.signal
```

Можно стартовать новую реплику.

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

Слот репликации инициализировался и используется:

```
| β=> SELECT * FROM pg_replication_slots \gx
```

```

-[ RECORD 1 ]-----+-----
slot_name      | replica
plugin         |
slot_type      | physical
datoid         |
database       |
temporary      | f
active         | t
active_pid     | 14881
xmin           |
catalog_xmin   |
restart_lsn    | 0/5037998
confirmed_flush_lsn |
wal_status     | reserved
safe_wal_size  |

```

Данные, измененные на новом мастере, получены:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5432 -d replica_switchover
```

```
α=> SELECT * FROM test;
```

```

          s
-----
Привет, мир!
Я - бывшая реплика (новый мастер).
(2 rows)

```

Проверим еще:

```
| β=> INSERT INTO test VALUES ('Еще строка с нового мастера.');
```

```
| INSERT 0 1
```

```
α=> SELECT * FROM test;
```

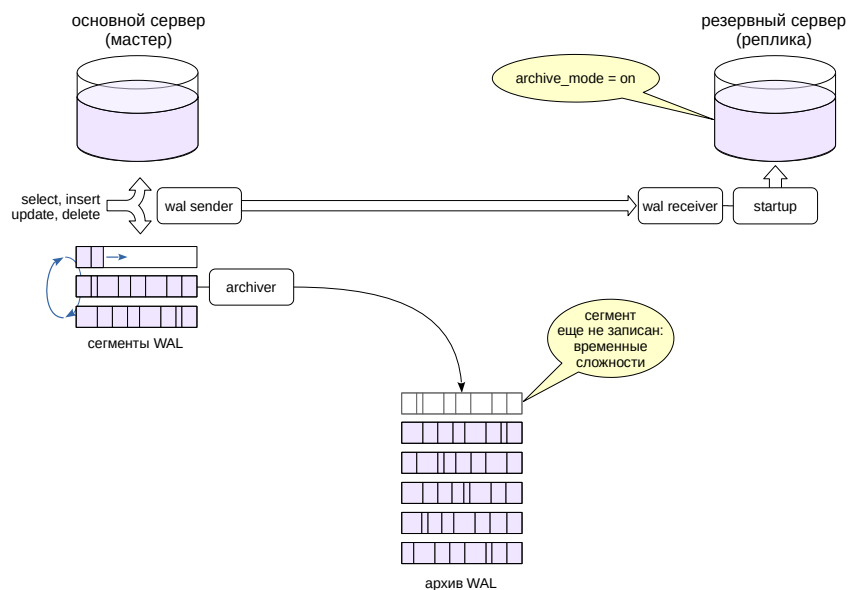
```

          s
-----
Привет, мир!
Я - бывшая реплика (новый мастер).
Еще строка с нового мастера.
(3 rows)

```

Таким образом, два сервера поменялись ролями.

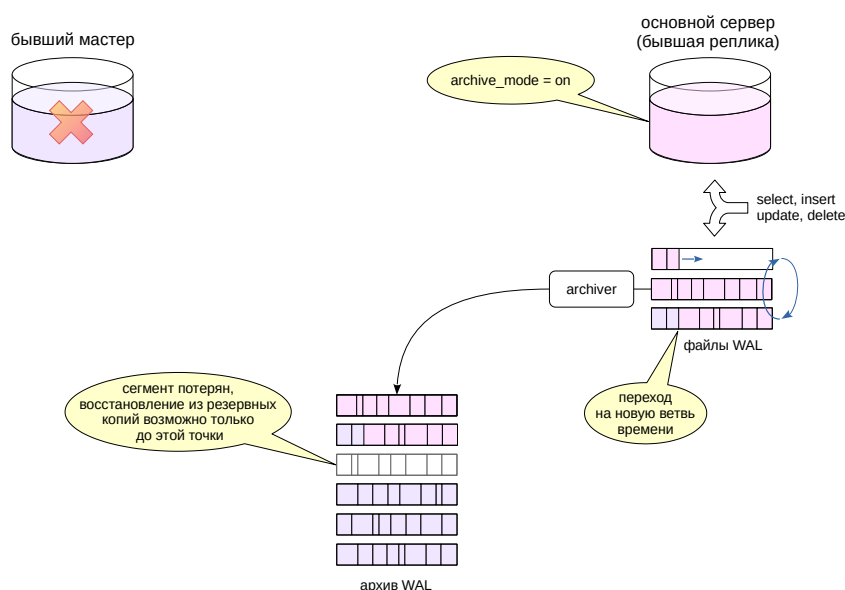
Переключение и архив



Архив файлов журнала предзаписи, наполняемый с помощью механизма непрерывного архивирования, имеет неприятную особенность в контексте потоковой репликации и переключении на реплику.

Допустим, при отказе мастера не все сегменты были записаны в архив. Например, могли возникнуть временные проблемы с доступным местом и `archive_command` возвращала ошибку.

Переключение и архив



13

Но реплика не в курсе настроек архивирования на мастере. Когда бывшая реплика займет место мастера, она не запишет недостающие сегменты в архив (хотя они у нее есть), потому что рассчитывает на то, что архив работал без сбоев. В результате архив будет неполным.

А это означает, что из имеющихся резервных копий можно восстановить систему только до образовавшейся «дыры». Если такая ситуация возникла (а это еще нужно понять), требуется в срочном порядке выполнить резервное копирование.

Потоковый архив лишен этого недостатка, поскольку утилита `pg_reseivewal` отслеживает содержимое каталога и запрашивает у сервера недостающие журнальные записи. Но, как отмечалось в модуле «Резервное копирование», использование этой утилиты сопряжено с поддержанием дополнительной инфраструктуры.

Проблемы с файловым архивом

Сейчас beta — основной сервер, а alpha — реплика. Настроим на обоих файловую архивацию в общий архив.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_switchover
```

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

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

```
| β=> \c - postgres
```

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

```
| β=> ALTER SYSTEM SET archive_mode = on;
```

```
| ALTER SYSTEM
```

```
| β=> ALTER SYSTEM SET archive_command = 'test ! -f /var/lib/postgresql/archive/%f && cp %p /var/lib/postgresql/archive/%f';
```

```
| ALTER SYSTEM
```

```
α=> \c - postgres
```

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

```
α=> ALTER SYSTEM SET archive_mode = on;
```

```
ALTER SYSTEM
```

```
α=> ALTER SYSTEM SET archive_command = 'test ! -f /var/lib/postgresql/archive/%f && cp %p /var/lib/postgresql/archive/%f';
```

```
ALTER SYSTEM
```

```
| β=> \q
```

```
α=> \q
```

Перезапускаем оба сервера.

```
student$ sudo pg_ctlcluster 13 beta restart
```

```
student$ sudo pg_ctlcluster 13 alpha restart
```

Текущий сегмент журнала:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_switchover -U postgres
```

```
| β=> SELECT pg_walfile_name(pg_current_wal_lsn());
```

```
|      pg_walfile_name  
|-----  
| 000000020000000000000005  
| (1 row)
```

Принудительно переключим сегмент WAL, выполнив какое-либо действие, а затем вызвав функцию pg_switch_wal.

```
| β=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();
```

```
| INSERT 0 1  
| pg_switch_wal  
|-----  
| 0/503A218  
| (1 row)
```

Сегмент попал в архив:

```
| β=> SELECT pg_walfile_name(pg_current_wal_lsn());
```

```
|      pg_walfile_name  
|-----  
| 000000020000000000000005  
| (1 row)
```

```
| β=> SELECT last_archived_wal, last_failed_wal FROM pg_catalog.pg_stat_archiver;
```

```
| last_archived_wal | last_failed_wal  
|-----+-----  
| 000000020000000000000005 |  
| (1 row)
```

Теперь представим, что возникли трудности с архивацией. Причиной может быть, например, заполнение диска или проблемы с сетевым соединением, а мы смоделируем их, возвращая статус 1 из команды архивации.

```
| β=> ALTER SYSTEM SET archive_command = 'exit 1';  
| SELECT pg_reload_conf();
```



```
ALTER SYSTEM
  pg_reload_conf
-----
t
(1 row)
```

Опять переключим сегмент WAL.

```
β=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();

INSERT 0 1
pg_switch_wal
-----
0/60000C0
(1 row)
```

Сегмент не архивируется.

```
β=> SELECT last_archived_wal, last_failed_wal FROM pg_catalog.pg_stat_archiver;

 last_archived_wal | last_failed_wal
-----+-----
000000020000000000000005 | 000000020000000000000006
(1 row)
```

Процесс archiver будет продолжать попытки, но безуспешно.

```
student$ tail -n 4 /var/log/postgresql/postgresql-13-beta.log
```

```
2022-05-20 14:08:58.599 MSK [15320] LOG:  received SIGHUP, reloading configuration files
2022-05-20 14:08:58.600 MSK [15320] LOG:  parameter "archive_command" changed to "exit 1"
2022-05-20 14:08:58.921 MSK [15326] LOG:  archive command failed with exit code 1
2022-05-20 14:08:58.921 MSK [15326] DETAIL:  The failed archive command was: exit 1
```

Alpha в режиме реплики не выполняла архивацию, а после перехода не будет архивировать пропущенный сегмент.

Остановим сервер beta, переключаемся на alpha.

```
β=> \q

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

15320

student$ sudo kill -9 15320

student$ sudo pg_ctlcluster 13 alpha promote
```

Еще раз принудительно переключим сегмент, теперь уже на alpha.

```
student$ /usr/lib/postgresql/13/bin/psql -U postgres -d replica_switchover

α=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();

INSERT 0 1
pg_switch_wal
-----
0/70022B0
(1 row)
```

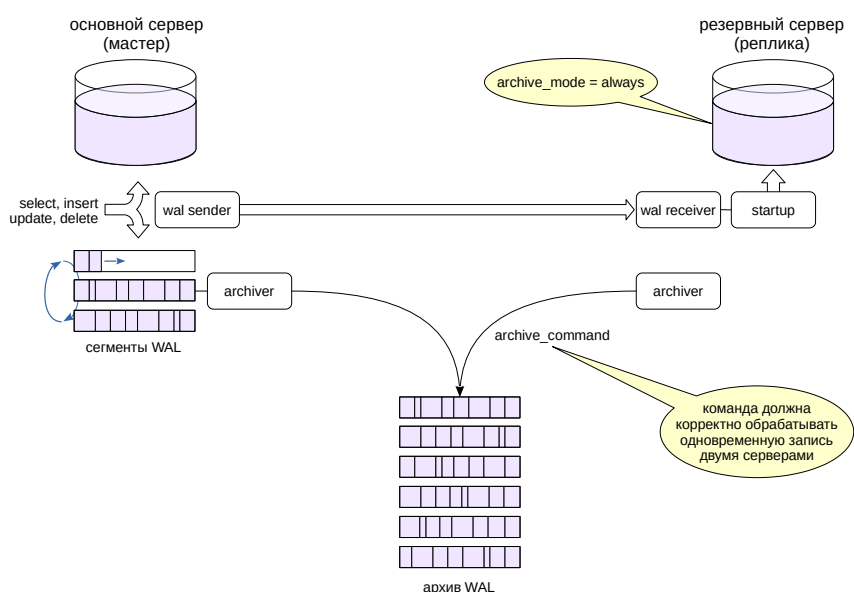
Что с архивом?

```
student$ ls -l /var/lib/postgresql/archive

total 32772
-rw----- 1 postgres postgres 16777216 мая 20 14:08 000000020000000000000005
-rw----- 1 postgres postgres 16777216 мая 20 14:09 000000030000000000000007
-rw----- 1 postgres postgres      83 мая 20 14:09 00000003.history
```

Сегмент отсутствует, архив теперь непригоден для восстановления и репликации.

Переключение и архив



15

При установке `archive_mode = always` на реплике запускается процесс `archiver`, который записывает сегменты в архив наравне с мастером. Таким образом, один и тот же файл будет записан два раза: и мастером, и репликой. Это накладывает на команду `archive_command` серьезные требования:

- она не должна перезаписывать существующий файл, но должна сообщать об успехе, если файл с тем же содержимым уже есть в архиве;
- она должна корректно обрабатывать одновременный вызов с двух серверов.

При такой настройке сегмент не пропадет, поскольку реплика будет продолжать попытки записи даже после останова мастера.

(Разумеется, можно настроить `archive_command` и так, чтобы мастер и реплика сохраняли сегменты журнала в разные архивы, но вряд ли это практично.)

Архивация с реплики

Чтобы при переключении на реплику архив не пострадал, на реплике нужно использовать значение `archive_mode = always`. При этом команда архивации должна корректно обрабатывать одновременную запись сегмента мастером и репликой.

Восстановим архивацию на сервере alpha. Команда архивации будет копировать файл, только если он отсутствует в архиве.

```
α=> ALTER SYSTEM SET archive_command = 'test -f /var/lib/postgresql/archive/%f || cp %p /var/lib/postgresql/archive/%f';
SELECT pg_reload_conf();
```

```
ALTER SYSTEM
  pg_reload_conf
-----
t
(1 row)
```

Добавим слот для реплики.

```
α=> SELECT pg_create_physical_replication_slot('replica');
```

```
pg_create_physical_replication_slot
-----
(replica,)
(1 row)
```

Теперь настроим beta как реплику с архивацией в режиме `always`.

```
student$ cat << EOF | sudo -u postgres tee /var/lib/postgresql/13/beta/postgresql.auto.conf
primary_conninfo='user=student port=5432'
primary_slot_name='replica'
archive_mode='always'
archive_command='test -f /var/lib/postgresql/archive/%f || cp %p /var/lib/postgresql/archive/%f'
EOF
```

```
primary_conninfo='user=student port=5432'
primary_slot_name='replica'
archive_mode='always'
archive_command='test -f /var/lib/postgresql/archive/%f || cp %p /var/lib/postgresql/archive/%f'
```

Стартуем реплику.

```
postgres$ touch /var/lib/postgresql/13/beta/standby.signal
```

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

Повторим опыт. Переключаем сегмент:

```
student$ /usr/lib/postgresql/13/bin/psql -U postgres -d replica_switchover
```

```
α=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();
```

```
INSERT 0 1
pg_switch_wal
-----
0/80001A8
(1 row)
```

Проверяем состояние архивации:

```
α=> SELECT last_archived_wal, last_failed_wal FROM pg_catalog.pg_stat_archiver;
```

```
last_archived_wal | last_failed_wal
-----+-----
000000030000000000000008 |
(1 row)
```

Заполненный сегмент попал в архив.

На сервере alpha возникли проблемы с архивацией, команда возвращает 1:

```
α=> ALTER SYSTEM SET archive_command = 'exit 1';
SELECT pg_reload_conf();
```

```
ALTER SYSTEM
  pg_reload_conf
-----
t
(1 row)
```

Alpha продолжает генерировать сегменты WAL.

```
α=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();
```

```
INSERT 0 1
pg_switch_wal
-----
0/900000C0
(1 row)
```

Но основной сервер их не архивирует:

```
α=> SELECT last_archived_wal, last_failed_wal FROM pg_catalog.pg_stat_archiver;
```

```
last_archived_wal | last_failed_wal
-----+-----
00000003000000000000000000000008 | 00000003000000000000000000000009
(1 row)
```

Однако архивация с реплики срабатывает, и сегмент оказывается в архиве:

```
student$ ls -l /var/lib/postgresql/archive
```

```
total 98308
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000100000000000000000005
-rw----- 1 postgres postgres 16777216 мая 20 14:08 0000000200000000000000000005
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000200000000000000000006
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000007
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000008
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000009
-rw----- 1 postgres postgres      83 мая 20 14:09 00000003.history
```

Выполняем переключение на реплику.

```
α=> \q
```

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

```
15367
```

```
student$ sudo kill -9 15367
```

```
student$ sudo pg_ctlcluster 13 beta promote
```

Beta стала основным сервером и генерирует файлы WAL.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -U postgres -d replica_switchover
```

```
| β=> INSERT INTO test SELECT now(); SELECT pg_switch_wal();
```

```
INSERT 0 1
pg_switch_wal
-----
0/A0028E0
(1 row)
```

Еще раз заглянем в архив:

```
student$ ls -l /var/lib/postgresql/archive
```

```
total 114696
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000100000000000000000005
-rw----- 1 postgres postgres 16777216 мая 20 14:08 0000000200000000000000000005
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000200000000000000000006
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000007
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000008
-rw----- 1 postgres postgres 16777216 мая 20 14:09 0000000300000000000000000009
-rw----- 1 postgres postgres      83 мая 20 14:09 00000003.history
-rw----- 1 postgres postgres 16777216 мая 20 14:09 000000040000000000000000000A
-rw----- 1 postgres postgres    125 мая 20 14:09 00000004.history
```

В архиве появились файлы реплики, пропусков в нем нет, проблема решена.

Переключение используется как в штатных,
так и в нештатных ситуациях

После переключения бывший мастер надо вернуть в строй

Обе процедуры должны быть заранее отработаны

Файловый архив журнала предзаписи требует внимания

1. Выполните необходимую настройку мастера и реплики для потоковой репликации с использованием слота, без непрерывного архивирования.
2. Имитируйте сбой основного сервера и переключитесь на реплику.
3. Верните в строй бывший основной сервер, выполнив резервную копию с нового мастера и настроив необходимые параметры.
Убедитесь, что репликация работает и использует слот.
4. Переключитесь на новую реплику, чтобы бывший мастер снова стал основным сервером.

2. Для имитации сбоя можно остановить сервер в режиме immediate:

```
pg_ctlcluster 13 имя_кластера stop -- -m immediate
```

или завершить основной процесс postgres:

```
kill -9 номер_процесса
```

Номер процесса можно найти в первой строке файла
PGDATA/postmaster.pid

1. Настройка репликации без архива

Настраиваем репликацию.

Создаем автономную резервную копию, предварительно создав слот.

```
student$ pg_basebackup --pgdata=/home/student/backup -R --slot=replica --create-slot
```

```
student$ sudo cat /home/student/backup/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=student passfile='/home/student/.pgpass' channel_binding=prefer host='/var/run/postgresql' port=5432 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
```

Выкладываем резервную копию в каталог PGDATA будущей реплики

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/beta
```

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

Запускаем сервер в режиме реплики.

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

Проверим настройки. Выполним несколько команд на мастере:

```
α=> CREATE DATABASE replica_switchover;
```

```
CREATE DATABASE
```

```
α=> \c replica_switchover
```

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

```
α=> CREATE TABLE test(s text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO test VALUES ('Привет, мир!');
```

```
INSERT 0 1
```

Проверим реплику:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433 -d replica_switchover
```

```
| β=> SELECT * FROM test;
```

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

2. Сбой основного сервера и переход на реплику

```
α=> \q
```

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

```
26382
```

```
student$ sudo kill -9 26382
```

```
student$ sudo pg_ctlcluster 13 beta promote
```

3. Возвращение в строй бывшего мастера

Создаем автономную резервную копию, предварительно создав слот.

Удалим конфигурационный файл, иначе basebackup его скопирует и запишет параметры.

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

```
student$ rm -rf /home/student/backup
```

```
student$ pg_basebackup -p 5433 --pgdata=/home/student/backup -R --slot=replica --create-slot
```

Параметры конфигурации и сигнальный файл подготовлены утилитой pg_basebackup:

```
student$ sudo cat /home/student/backup/postgresql.auto.conf
```

```
primary_conninfo = 'user=student passfile='/home/student/.pgpass' channel_binding=prefer host='/var/run/postgresql' port=5433 sslmode=prefer sslcompression=0 sslsni=1 ssl_min_proto
primary_slot_name = 'replica'
```

```
student$ ls -l /home/student/backup/standby.signal
```

```
-rw----- 1 student student 0 мая 20 14:16 /home/student/backup/standby.signal
```

Выкладываем копию на бывший мастер и запускаем новую реплику.

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

```
student$ sudo mv /home/student/backup /var/lib/postgresql/13/alpha
```

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

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

Слот репликации инициализировался и используется:

```
| β=> SELECT * FROM pg_replication_slots \gx
```

```
|-[ RECORD 1 ]-----+-----|
| slot_name         | replica |
| plugin            |         |
| slot_type         | physical |
| datoid            |         |
| database          |         |
| temporary         | f       |
| active            | t       |
| active_pid        | 26969   |
| xmin              |         |
| catalog_xmin      |         |
| restart_lsn       | 0/7000060 |
| confirmed_flush_lsn |         |
| wal_status        | reserved |
| safe_wal_size     |         |
```

Проверим еще:

```
| β=> INSERT INTO test VALUES ('Я - бывшая реплика (новый мастер).');
```

```
| INSERT 0 1
```

```
student$ /usr/lib/postgresql/13/bin/psql -p 5432 -d replica_switchover
```

```
α=> SELECT * FROM test;
```

```
-----  
Привет, мир!  
Я - бывшая реплика (новый мастер).  
(2 rows)
```

4. Переключение на новую реплику

```
student$ sudo pg_ctlcluster 13 alpha promote  
a=> select pg_is_in_recovery();  
  
pg_is_in_recovery  
-----  
f  
(1 row)
```

В итоге прежний мастер снова стал основным сервером.