

Replication Switchover to Replica



Copyright

© Postgres Professional, 2017–2025

Authors: Egor Rogov, Pavel Luzanov, Ilya Bashtanov, Alexey Beresnev

Photo: Oleg Bartunov (Phu Monastery and Bhrikuti Peak, Nepal)

Use of Course Materials

Non-commercial use of course materials (presentations, demonstrations) is allowed without restrictions. Commercial use is possible only with the written permission of Postgres Professional. It is prohibited to make changes to the course materials.

Feedback

Please send your feedback, comments and suggestions to:

edu@postgrespro.ru

Disclaimer

Postgres Professional assumes no responsibility for any damages and losses, including loss of income, caused by direct or indirect, intentional or accidental use of course materials. Postgres Professional company specifically disclaims any warranties on course materials. Course materials are provided “as is,” and Postgres Professional company has no obligations to provide maintenance, support, updates, enhancements, or modifications.

Switchover to Replica

Failback on Former Leader

Considerations for File-Based Archiving

Switchover to Replica

Reasons

- planned switchover:
scheduled maintenance
- emergency switchover (failover):
switch to replica after a leader failure

Procedure

- make sure the leader is stopped
- `$ pg_ctl promote` or
`pg_promote()`
- automatic switchover: unavailable
(third-party cluster management software required)

You might need to switch to a standby server for maintenance work on the primary server. In this case, the switchover can be performed at a convenient time in a planned manner.

In the event of a main server failure, you need to switch to the standby server as quickly as possible to minimize system downtime (failover).

A replica is promoted to independent mode by either executing the command `pg_ctl promote` from the OS or the function `pg_promote` from SQL. Switching with the trigger file (the *`promote_trigger_file` option*) is no longer supported in PostgreSQL 16 and higher.

Before promoting a replica, you must make sure that the original leader is stopped. If it continues to work and clients are still connected to it, data on different servers can disperse, and reconciling them afterwards is messy, non-automatable, and often resulting in some data being lost.

To switch to a standby server, you must redirect clients to it. This requires the use of third-party software (as discussed in the “Cluster Technologies” module), since PostgreSQL does not have its own tools for this.

<https://postgrespro.com/docs/postgresql/16/warm-standby-failover>

Replica Promotion



Apply WAL records

already received by the wal receiver but not yet applied by the startup process

Terminate processes

wal receiver
startup

Start processes

wal writer
autovacuum launcher
archiver (if configured)

Switch to a new timeline

4

After receiving the signal, the replica gets promoted: it performs recovery and becomes a regular server.

The startup process applies log records that have already been received by the wal receiver process, but have not yet been applied. Then, the wal receiver and startup processes terminate since they are not needed on the leader.

On the other hand, the wal writer and autovacuum launcher processes are started up (archiver too, if configured.)

After completing recovery, the server switches to a new timeline.

Tiny details aside, the process is essentially just a recovery from backup.

Streaming Replication Configuration

In this demo, we will run checkpoints manually and increase the scheduled interval.

```
α=> ALTER SYSTEM SET checkpoint_timeout = '1d';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 alpha reload
```

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

```
CREATE DATABASE
```

Let's enable the `pg_walinspect` extension. It will help us inspect checkpoint entries in the WAL. Superuser privileges are required.

```
α=> \c replica_switchover postgres
```

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

```
α=> CREATE EXTENSION pg_walinspect;
```

```
CREATE EXTENSION
```

```
α=> \c - student
```

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

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

```
CREATE TABLE
```

```
α=> INSERT INTO test VALUES ('Initial state. The only server is alpha.');
```

```
INSERT 0 1
```

We will be using the standalone backup multiple times, so we create it in tar format.

```
student$ pg_basebackup \
--format=tar \
--compress=gzip \
--checkpoint=fast \
--pgdata=/home/student/tmp/backup
```

Let's place the backup into the PGDATA directory on the beta server:

```
student$ sudo pg_ctlcluster 16 beta stop
```

Cluster is not running.

```
postgres$ rm -rf /var/lib/postgresql/16/beta
```

```
postgres$ mkdir /var/lib/postgresql/16/beta
```

```
postgres$ chmod 700 /var/lib/postgresql/16/beta
```

```
student$ sudo tar xzf /home/student/tmp/backup/base.tar.gz -C /var/lib/postgresql/16/beta
```

```
student$ sudo tar xzf /home/student/tmp/backup/pg_wal.tar.gz -C /var/lib/postgresql/16/beta/pg_wal
```

We will need the standby.signal file.

```
postgres$ touch /var/lib/postgresql/16/beta/standby.signal
```

Configure the connection to the primary.

```
postgres$ cat << EOF >> /var/lib/postgresql/16/beta/postgresql.auto.conf
primary_conninfo = 'user=student port=5432'
primary_slot_name = 'replica'
EOF
```

Create a replication slot on alpha and start the replica beta.

```
student$ pg_receivewal --create-slot --slot=replica
```

```
student$ sudo pg_ctlcluster 16 beta start
```

Check the replica:

```
α=> INSERT INTO test VALUES ('Replication check: alpha – primary, beta – replica.');
```

```
INSERT 0 1
```

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

```
| β=> SELECT * FROM test;

          s
-----
Initial state. The only server is alpha.
Replication check: alpha – primary, beta – replica.
(2 rows)
```

Stopping the Primary and Creating the Checkpoint

Let's add one more row to the table and gracefully stop the primary. A checkpoint will be performed in the process.

```
α=> INSERT INTO test VALUES ('Before graceful stop of primary alpha.');
```

```
INSERT 0 1
```

```
student$ sudo pg_ctlcluster 16 alpha stop
```

Checkpoint information has been written to the control file:

```
postgres$ /usr/lib/postgresql/16/bin/pg_controldata -D ~postgres/16/alpha | \
grep -E '^(cluster|checkpoint.*location).*:'
```

```
Database cluster state:          shut down
Latest checkpoint location:      0/3000200
Latest checkpoint's REDO location: 0/3000200
```

Did the replica receive this checkpoint record?

Let's examine the WAL on the replica, starting from the Log Sequence Number of the checkpoint record.

```
| β=> SELECT start_lsn, prev_lsn, resource_manager, record_type, record_length,
         left(description, 46) || '...' description
FROM pg_get_wal_records_info('0/3000200', 'FFFFFFFF/FFFFFFFF') \gx

-[ RECORD 1 ]-----+-----
start_lsn      | 0/3000200
prev_lsn       | 0/30001D8
resource_manager | XLOG
record_type    | CHECKPOINT_SHUTDOWN
record_length  | 114
description    | redo 0/3000200; tli 1; prev tli 1; fpw true; x...
```

The record_length field shows the record length in bytes.

The last position received by the replica is greater than the LSN of the checkpoint record:

```
| β=> SELECT pg_last_wal_receive_lsn(), pg_wal_lsn_diff(pg_last_wal_receive_lsn(), '0/3000200' );

 pg_last_wal_receive_lsn | pg_wal_lsn_diff
-----+-----
0/3000278                |                120
(1 row)
```

The difference slightly exceeds the record length (114) due to WAL record alignment: on a 64-bit platform, record LSNs are multiples of eight bytes.

Replica Promotion

The beta server is currently a replica (it is in recovery mode):

```
| β=> SELECT pg_is_in_recovery();

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

Promote the replica. You can also use the `pg_promote()` function, which does the same thing.

```
student$ sudo pg_ctlcluster 16 beta promote
```

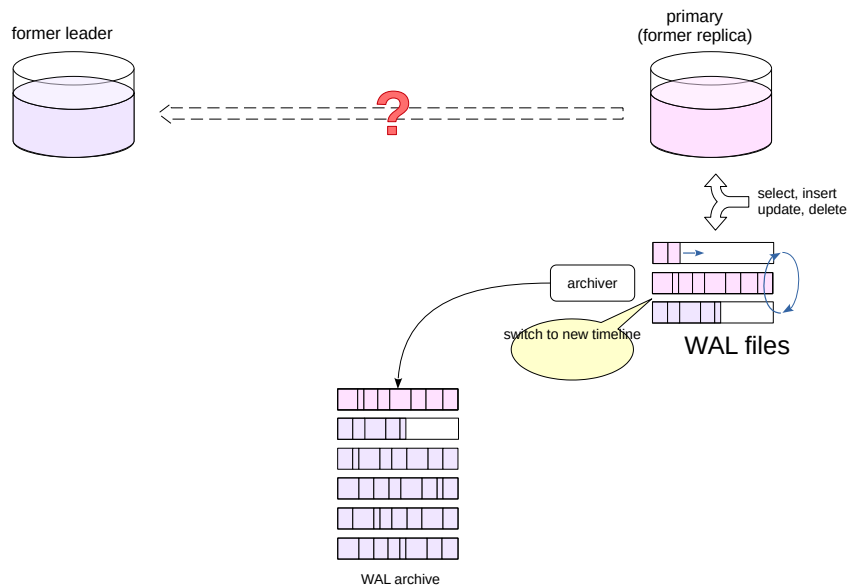
There is another way to promote a replica: delete the standby.signal file and restart the standby server. This is not quite a "proper" way to do it, since the replica won't recognize that recovery is complete and won't switch to a new timeline.

The former replica beta is now a fully functional server instance.

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

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

Failback

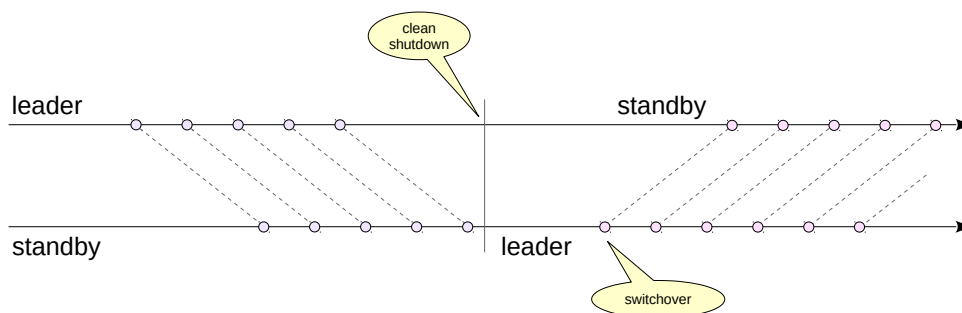


After promotion, the former replica acts as a standalone server. It processes client queries and generates its own WAL archive – within a new timeline.

If the switchover wasn't caused by a serious server failure, you'll need to perform a failback: quickly bring that server back online – this time as a replica.

Simple Reconnect

The former leader connects to the new leader as a replica
works only if the replica received all WAL records from the former leader before the switchover



7

When the leader is shut down cleanly (using the fast or the smart shutdown mode), all of its WAL records will most likely reach the replica, though this isn't guaranteed.

The shutdown procedure first terminates all backend processes, then performs a checkpoint, and only at the very end stops wal sender, giving the replica time to receive the checkpoint record.

You can check the leader's position in WAL with `pg_controldata` (look for "latest checkpoint location"), while the replica's position is shown by the `pg_last_wal_receive_lsn` function. This function returns the *current* position in WAL, so its value must be *ahead of* the latest checkpoint location by the size of the checkpoint record.

If all records were actually received, then the former leader can immediately connect to the new one just by adjusting the configuration accordingly.

If the leader is stopped without performing a checkpoint (due to a crash or shutdown in the immediate mode), such a connection will likely be impossible on a server with a current load. The server won't be able to start, and an error will be logged.

Rejoining the Cluster: Direct Connection

Bring the alpha server back online, but this time as a replica. Create the standby.signal file:

```
postgres$ touch /var/lib/postgresql/16/alpha/standby.signal
```

Configure the connection to the primary.

```
postgres$ cat << EOF >> /var/lib/postgresql/16/alpha/postgresql.auto.conf
primary_conninfo = 'user=student port=5433'
EOF
```

Start alpha and verify that it has become a replica.

```
student$ sudo pg_ctlcluster 16 alpha start
```

```
student$ psql -d replica_switchover
```

```
α=> SELECT pg_is_in_recovery();
```

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

On the beta server, we can modify data:

```
| β=> INSERT INTO test VALUES ('The beta server is now the primary, and alpha is running as a replica.');
```

```
| INSERT 0 1
```

The alpha replica receives them.

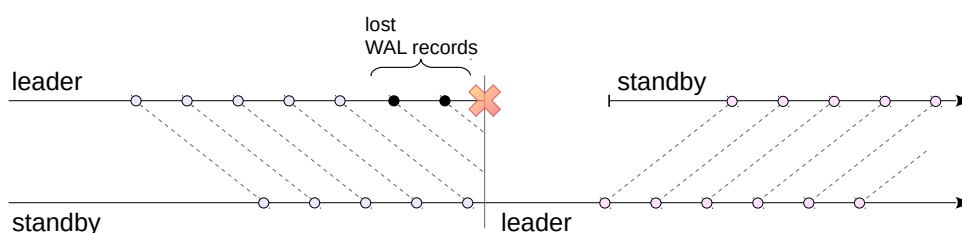
```
α=> SELECT * FROM test;
```

```

s
-----
Initial state. The only server is alpha.
Replication check: alpha – primary, beta – replica.
Before graceful stop of primary alpha.
The beta server is now the primary, and alpha is running as a replica.
(4 rows)
```

The former leader is restored from a backup

creates a completely new replica, takes a long time
can be expedited with rsync if not much time have passed
(still takes a while for large databases)



When a leader crashes, there's a good chance that some WAL records don't make it to the replica. You can't just reconnect the former leader as a replica in this case.

The simplest and most reliable approach is to build a completely new replica by creating and deploying a fresh base backup. However, for large databases, this can take a long time.

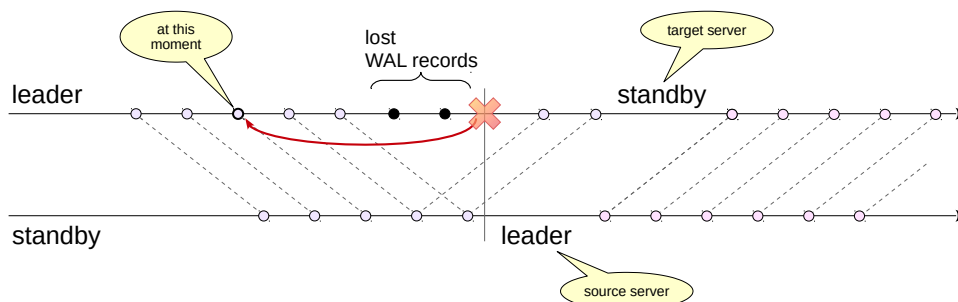
A variation of this approach is to skip `pg_basebackup` and instead create the backup using the backup API and `rsync`.

If you create the backup immediately after promoting the replica, most files won't have changed yet, which can significantly speed up the process.

But naturally, this does complicate things.

Preparing for Recovery

revert the lost WAL records by replacing the affected pages on the target server with pages from the source server
copy all system files from the source server



10

An even faster approach is to use the stock tool `pg_rewind`.

It identifies where the two servers diverged, finds the nearest common checkpoint before that point, and by scanning the WAL, determines which pages have been modified since then.

These pages are then replaced with their counterparts from the source server (the new leader), and there shouldn't be many of them. The utility also copies all system files from the source server.

The server then applies all necessary WAL records from the new leader. At this point, it's no longer the `pg_rewind` doing the work, but rather the standard recovery process that runs when the server starts up. To ensure recovery starts from the correct point, the utility creates a `backup_label` control file.

<https://postgrespro.com/docs/postgresql/16/app-pgrewind>

Considerations with pg_rewind

Target server (former leader)

all necessary WAL files must be either in `pg_wal` or in the archive

checksums must be enabled or `wal_log_hints` must be on
the parameter `full_page_writes` must be on

Source server (current leader)

client-server connection:

`--source-server` = connection string

cleanly shut down server:

`--source-pgdata` = directory

For the source server, if it is running, the `--source-server` switch specifies the libpq client-server connection string (the utility does not use the replication protocol). The connecting role must have the privileges required to run `pg_ls_dir`, `pg_stat_file`, and `pg_read_binary_file` functions.

If the source server was shut down cleanly, you can just point to its data directory with the `--source-pgdata` switch.

Several specific requirements must be met for the tool to work:

- All WAL segments containing records from the last common checkpoint to the current moment must be either in the target server's `pg_wal` directory or present and available from the WAL archive.
In the second case, you need to set the *restore_command* parameter and the `--restore-target-wal` switch.
- Due to errors during writing, data mismatch within the pages is possible (the non-atomic writing problem, discussed in the "Base backup" module). This is solved by having the target server write each page's full image to WAL after it is first changed since the checkpoint. This is enabled by *full_page_writes* = on. This mode does ignore "minor" page changes (hint bits), so the cluster also must either have page checksum calculation enabled or have *wal_log_hints* = on.
- If the target server was shut down incorrectly and without a checkpoint, the tool will start it in single-user mode and immediately perform a clean shutdown.

Rejoining the Cluster: pg_rewind

Let's now simulate a split-brain scenario – two writable servers resulting from a network partition (we cover this and other types of failures in detail in the "Cluster Technologies Overview" topic). Promote the alpha server out of replica mode without stopping the primary beta server.

```
student$ sudo pg_ctlcluster 16 alpha promote
```

Both alpha and beta can now accept write operations:

```
| β=> INSERT INTO test VALUES ('Die hard');
|
| INSERT 0 1
```

In practice, this situation must be avoided at all costs, since it's no longer clear which server to trust. You'll have to either discard all changes on one of the servers or figure out how to merge the data.

We choose to discard the changes made on the beta server.

Roll back the conflicting changes on beta – those that diverge from the changes made on alpha. To do this, we'll use the `pg_rewind` utility – so let's first verify that the required parameter values are set on the target server (beta):

```
| β=> \dconfig data_checksums|full_page_writes

List of configuration parameters
Parameter | Value
-----+-----
data_checksums | on
full_page_writes | on
(2 rows)
```

Stop the target server ungracefully.

```
student$ sudo kill -QUIT $(sudo head -n 1 /var/lib/postgresql/16/beta/postmaster.pid)
```

The `pg_rewind` utility requires you to specify the target server's PGDATA directory and how to connect to the source server (alpha): either a connection string (if the server is running) or the path to its PGDATA directory (if it is stopped).

In our case, the alpha server is running.

Since the target server beta was not shut down properly, the utility will start it in single-user mode and stop it after performing a checkpoint. To start it, the configuration file location must be provided.

The `-R` option ensures the target server starts as a replica once the utility finishes: after consistency is restored, it will remain in recovery mode.

```
postgres$ /usr/lib/postgresql/16/bin/pg_rewind \
-D /var/lib/postgresql/16/beta \
--config-file=/etc/postgresql/16/beta/postgresql.conf \
-R \
--source-server='user=student port=5432'
```

```
pg_rewind: executing "/usr/lib/postgresql/16/bin/postgres" for target server to complete
crash recovery
2026-09-16 23:59:46.566 MSK [78455] LOG:  database system was interrupted; last known up
at 2026-09-16 23:59:38 MSK
2026-09-16 23:59:46.567 MSK [78455] LOG:  database system was not properly shut down;
automatic recovery in progress
2026-09-16 23:59:46.569 MSK [78455] LOG:  redo starts at 0/30002A8
2026-09-16 23:59:46.570 MSK [78455] LOG:  invalid record length at 0/3006AA0: expected at
least 24, got 0
2026-09-16 23:59:46.570 MSK [78455] LOG:  redo done at 0/3006A60 system usage: CPU: user:
0.00 s, system: 0.00 s, elapsed: 0.00 s
2026-09-16 23:59:46.572 MSK [78455] LOG:  checkpoint starting: end-of-recovery immediate
wait
2026-09-16 23:59:46.573 MSK [78455] LOG:  checkpoint complete: wrote 9 buffers (0.1%); 0
WAL file(s) added, 0 removed, 0 recycled; write=0.001 s, sync=0.001 s, total=0.002 s;
sync files=0, longest=0.000 s, average=0.000 s; distance=25 kB, estimate=25 kB;
lsn=0/3006AA0, redo lsn=0/3006AA0
```

```
PostgreSQL stand-alone backend 16.14 (Ubuntu 16.14-1.pgdg24.04+1)
backend> 2026-09-16 23:59:46.581 MSK [78455] LOG:  checkpoint starting: shutdown immediate
2026-09-16 23:59:46.581 MSK [78455] LOG:  checkpoint complete: wrote 0 buffers (0.0%); 0
WAL file(s) added, 0 removed, 0 recycled; write=0.001 s, sync=0.001 s, total=0.001 s;
sync files=0, longest=0.000 s, average=0.000 s; distance=0 kB, estimate=23 kB;
lsn=0/3006B18, redo lsn=0/3006B18
```

```
pg_rewind: servers diverged at WAL location 0/3002AE8 on timeline 2
pg_rewind: rewinding from last common checkpoint at 0/30028B0 on timeline 2
pg_rewind: Done!
```

As a result, pg_rewind "rolled back" the data pages to the last checkpoint before the point where the two servers diverged.

The utility also created a backup_label file containing the WAL position from which recovery should begin. Let's take a look inside:

```
postgres$ cat /var/lib/postgresql/16/beta/backup_label

START WAL LOCATION: 0/30002A8 (file 000000020000000000000003)
CHECKPOINT LOCATION: 0/30028B0
BACKUP METHOD: pg_rewind
BACKUP FROM: standby
START TIME: 2026-09-16 23:59:47 MSK
```

By using the -R option, we instructed the utility to create the standby.signal file and set the connection string in the configuration file.

```
postgres$ ls -l /var/lib/postgresql/16/beta/standby.signal

-rw----- 1 postgres postgres 0 Sep 16 23:59 /var/lib/postgresql/16/beta/standby.signal
```

A replication slot has already been created on the alpha server – let's make use of it.

```
postgres$ echo primary_slot_name = 'replica' >> /var/lib/postgresql/16/beta/postgresql.auto.conf
```

```
postgres$ cat /var/lib/postgresql/16/beta/postgresql.auto.conf

# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
checkpoint_timeout = '1d'
primary_conninfo = 'user=student port=5433'
primary_conninfo = 'user=student passfile='/var/lib/postgresql/.pgpass''
channel_binding=prefer port=5432 sslmode=prefer sslnegotiation=postgres sslcompression=0
sslcertmode=allow sslsn=1 ssl_min_protocol_version=TLSv1.2 gssencmode=prefer
krbsrvname=postgres gssdelegation=0 target_session_attrs=any load_balance_hosts=disable'
primary_slot_name = replica
```

Now beta will start up as a replica.

```
student$ sudo pg_ctlcluster 16 beta start
```

The replication slot has been initialized and is now in use:

```
α=> SELECT * FROM pg_replication_slots \gx
```

```
-[ RECORD 1 ]-----+-----
slot_name      | replica
plugin         |
slot_type      | physical
datoid         |
database       |
temporary      | f
active         | t
active_pid     | 78737
xmin           |
catalog_xmin   |
restart_lsn    | 0/305E000
confirmed_flush_lsn |
wal_status     | reserved
safe_wal_size  |
two_phase     | f
conflicting    |
```

Data modified on the new primary (alpha) has been received by the replica (beta):

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

```
β=> SELECT * FROM test;

          s
-----
Initial state. The only server is alpha.
Replication check: alpha – primary, beta – replica.
Before graceful stop of primary alpha.
The beta server is now the primary, and alpha is running as a replica.
(4 rows)
```

Let's add one more row:

```
α=> INSERT INTO test VALUES ('Another row from alpha.');
```

INSERT 0 1

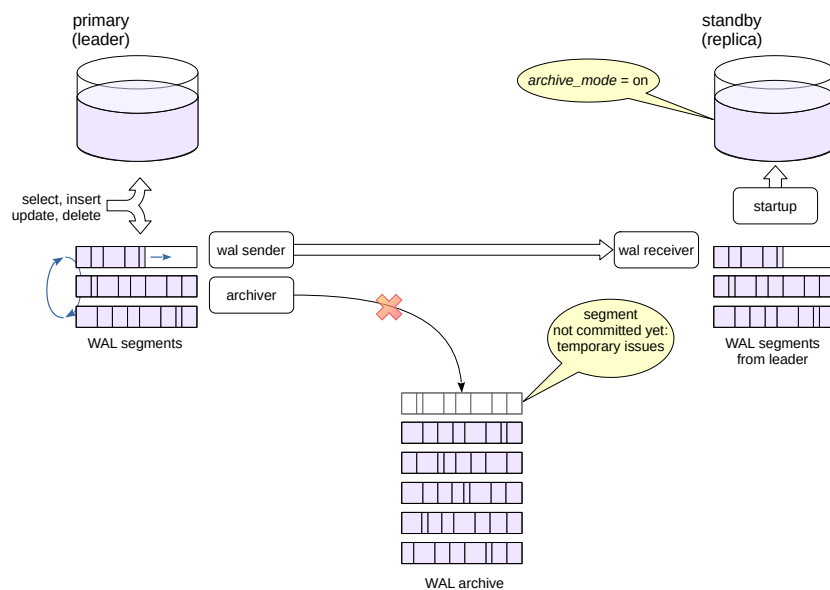
β=> SELECT * FROM test;

s

Initial state. The only server is alpha.
Replication check: alpha – primary, beta – replica.
Before graceful stop of primary alpha.
The beta server is now the primary, and alpha is running as a replica.
Another row from alpha.
(5 rows)

Now the former primary (beta) is back into service as a replica.

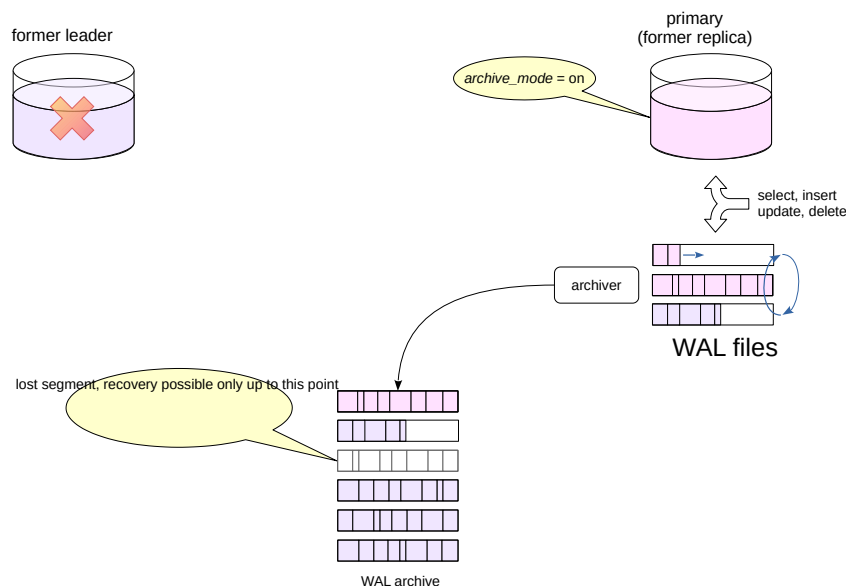
Switchover and Archiving



The write-ahead log archive, populated through continuous archiving, has an problem when it comes to streaming replication and switchover:

on a leader failure, some WAL segments may not make it into the archive. For example, if the disk runs out of space, `archive_command` will return an error.

Switchover and Archiving



The replica is unaware of the archiving settings on the leader. When the former replica takes over as the leader, it won't write the missing file segments to the archive (even though it could) because it assumes the archive was functioning properly all along. The archive will end up incomplete.

This means you can only restore the system from existing backups up to the point where this gap occurred. If this situation has occurred (and you detect it in time), you must take a backup immediately.

Stream archiving doesn't have this problem because the `pg_receivewal` utility monitors the directory contents and requests any missing WAL records from the server.

However, as mentioned in the "Backup" module, using this tool means maintaining additional infrastructure.

File Archive Issues

Let's restore the alpha and beta servers from a backup and configure file archiving to a shared archive on both.

```
student$ sudo pg_ctlcluster 16 alpha stop
student$ sudo pg_ctlcluster 16 beta stop
postgres$ rm -rf /var/lib/postgresql/16/alpha
postgres$ mkdir /var/lib/postgresql/16/alpha
postgres$ chmod 700 /var/lib/postgresql/16/alpha
postgres$ rm -rf /var/lib/postgresql/16/beta
postgres$ mkdir /var/lib/postgresql/16/beta
postgres$ chmod 700 /var/lib/postgresql/16/beta
student$ sudo tar xzf /home/student/tmp/backup/base.tar.gz -C /var/lib/postgresql/16/alpha
student$ sudo tar xzf /home/student/tmp/backup/pg_wal.tar.gz -C /var/lib/postgresql/16/alpha/pg_wal
student$ sudo tar xzf /home/student/tmp/backup/base.tar.gz -C /var/lib/postgresql/16/beta
student$ sudo tar xzf /home/student/tmp/backup/pg_wal.tar.gz -C /var/lib/postgresql/16/beta/pg_wal
```

WAL segment archive.

```
postgres$ rm -rf /var/lib/postgresql/archive
postgres$ mkdir /var/lib/postgresql/archive
```

Identical archiving settings on both the alpha and beta servers.

```
postgres$ cat << EOF | tee -a \
    /var/lib/postgresql/16/alpha/postgresql.auto.conf \
    /var/lib/postgresql/16/beta/postgresql.auto.conf
archive_mode = on
archive_command = 'test ! -f /var/lib/postgresql/archive/%f && cp %p /var/lib/postgresql/archive/%f'
EOF

archive_mode = on
archive_command = 'test ! -f /var/lib/postgresql/archive/%f && cp %p
/var/lib/postgresql/archive/%f'
```

On the replica (beta), let's create the standby.signal file and configure the connection to the primary.

```
postgres$ touch /var/lib/postgresql/16/beta/standby.signal

postgres$ cat << EOF >> /var/lib/postgresql/16/beta/postgresql.auto.conf
primary_conninfo = 'user=student port=5432'
primary_slot_name = 'replica'
EOF
```

Start the primary (alpha) and create a replication slot.

```
student$ sudo pg_ctlcluster 16 alpha start
student$ pg_receivewal --create-slot --slot=replica
```

Start the replica.

```
student$ sudo pg_ctlcluster 16 beta start
```

Current WAL segment:

```
student$ psql -d replica_switchover
α=> SELECT pg_walfile_name(pg_current_wal_lsn());

   pg_walfile_name
-----
000000010000000000000003
(1 row)
```

Force a WAL segment switch. For the switch to occur, we need to ensure that both the current and next segments contain some records.

```
α=> SELECT pg_switch_wal();
```

```

pg_switch_wal
-----
0/3006658
(1 row)

```

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

```
INSERT 0 1
```

Now log entries are being written to the next segment, while the previous one has been archived:

```

α=> SELECT
    pg_walfile_name(pg_current_wal_lsn()) current_wal,
    last_archived_wal,
    last_failed_wal
FROM pg_stat_archiver;

```

current_wal	last_archived_wal	last_failed_wal
000000010000000000000004	000000010000000000000003	

```

(1 row)

```

Let's say there is an issue with archiving. The cause could be something like a running out of space on disk, or network connectivity issues – we'll simulate this by returning status 1 from the archive command.

```
α=> ALTER SYSTEM SET archive_command = 'exit 1';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 alpha reload
```

Let's force another WAL segment switch.

```
α=> SELECT pg_switch_wal();
```

```

pg_switch_wal
-----
0/4002BE0
(1 row)

```

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

```
INSERT 0 1
```

The segment is not being archived.

```

α=> SELECT
    pg_walfile_name(pg_current_wal_lsn()) current_wal,
    last_archived_wal,
    last_failed_wal
FROM pg_stat_archiver;

```

current_wal	last_archived_wal	last_failed_wal
000000010000000000000005	000000010000000000000003	000000010000000000000004

```

(1 row)

```

The archiver process will keep retrying, but without success.

```
student$ tail -n 4 /var/log/postgresql/postgresql-16-alpha.log
```

```

2026-09-17 00:01:06.202 MSK [79509] LOG:  archive command failed with exit code 1
2026-09-17 00:01:06.202 MSK [79509] DETAIL:  The failed archive command was: exit 1
2026-09-17 00:01:07.204 MSK [79509] LOG:  archive command failed with exit code 1
2026-09-17 00:01:07.204 MSK [79509] DETAIL:  The failed archive command was: exit 1

```

The beta server wasn't archiving while in replica mode, and after promotion it won't archive the missed segment either.

Stop alpha and promote beta.

```
student$ sudo kill -QUIT $(sudo head -n 1 /var/lib/postgresql/16/alpha/postmaster.pid)
```

```
student$ sudo pg_ctlcluster 16 beta promote
```

Force another WAL segment switch, this time on beta.

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

```
| β=> SELECT pg_switch_wal();
```

```
pg_switch_wal
-----
0/5003E50
(1 row)
```

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

```
INSERT 0 1
```

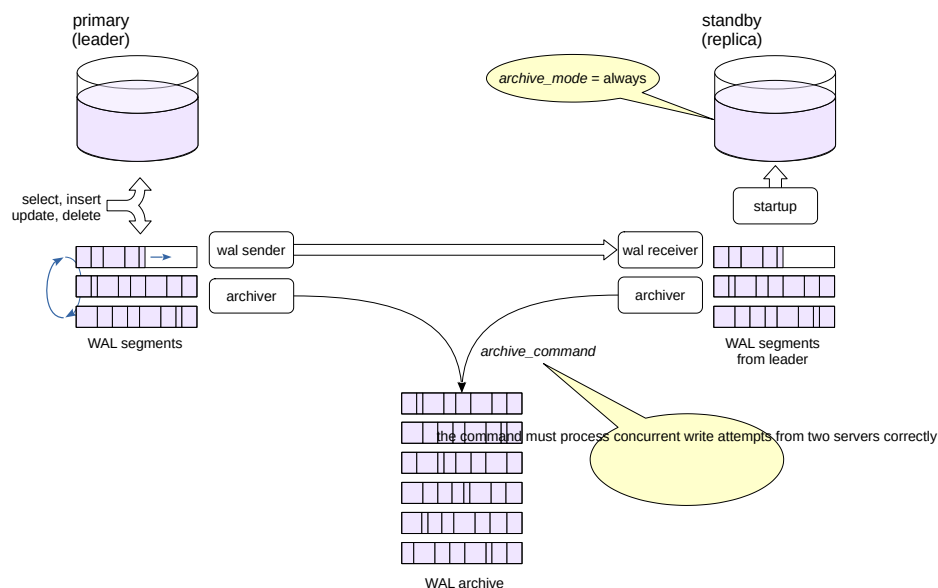
How's the archive doing?

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

```
total 49156
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000001000000000000000003
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000001000000000000000005.partial
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000002000000000000000005
-rw----- 1 postgres postgres      41 Sep 17 00:01 00000002.history
```

Segment 00000001000000000000000004 is missing – the archive is now unusable for recovery and replication.

Switchover and Archiving



16

With `archive_mode = always`, the replica starts an archiver process that writes segments to the archive just like the leader server does. This means that the same file will be recorded twice, by two servers. This places two strict requirements on `archive_command`:

- it must not overwrite an existing file, but should report success if a file with identical content is already present in the archive;
- it must handle concurrent write attempts from both servers correctly.

With these requirements met, the segment won't be lost because the replica will keep trying to write it even after the leader server stops.

Of course, you could configure `archive_command` so that the leader and the replica save WAL segments to separate archives, but that's hardly practical.

Archiving from a Replica

To keep the archive intact when failing over to a replica, the replica must have `archive_mode = always`. The archive command must correctly handle the case where both the primary and the replica write the same segment simultaneously.

Let's restore both alpha and beta from backup again and configure file-based archiving to a shared archive on both servers.

```
student$ sudo pg_ctlcluster 16 alpha stop
```

Cluster is not running.

```
student$ sudo pg_ctlcluster 16 beta stop
```

```
postgres$ rm -rf /var/lib/postgresql/16/alpha
```

```
postgres$ mkdir /var/lib/postgresql/16/alpha
```

```
postgres$ chmod 700 /var/lib/postgresql/16/alpha
```

```
postgres$ rm -rf /var/lib/postgresql/16/beta
```

```
postgres$ mkdir /var/lib/postgresql/16/beta
```

```
postgres$ chmod 700 /var/lib/postgresql/16/beta
```

```
student$ sudo tar xzf /home/student/tmp/backup/base.tar.gz -C /var/lib/postgresql/16/alpha
```

```
student$ sudo tar xzf /home/student/tmp/backup/pg_wal.tar.gz -C /var/lib/postgresql/16/alpha/pg_wal
```

```
student$ sudo tar xzf /home/student/tmp/backup/base.tar.gz -C /var/lib/postgresql/16/beta
```

```
student$ sudo tar xzf /home/student/tmp/backup/pg_wal.tar.gz -C /var/lib/postgresql/16/beta/pg_wal
```

WAL segment archive.

```
postgres$ rm -rf /var/lib/postgresql/archive
```

```
postgres$ mkdir /var/lib/postgresql/archive
```

Configuration of archiving on the primary (alpha). We replaced the logical operator in the archive command so that it immediately returns 0 if the file already exists.

```
postgres$ cat << EOF | tee -a /var/lib/postgresql/16/alpha/postgresql.auto.conf
archive_mode = on
archive_command = 'test -f /var/lib/postgresql/archive/%f || cp %p /var/lib/postgresql/archive/%f'
EOF
```

```
archive_mode = on
archive_command = 'test -f /var/lib/postgresql/archive/%f || cp %p
/var/lib/postgresql/archive/%f'
```

Configuration of archiving on the replica (beta). Use the always mode and the same archive command.

```
postgres$ cat << EOF | tee -a /var/lib/postgresql/16/beta/postgresql.auto.conf
archive_mode = always
archive_command = 'test -f /var/lib/postgresql/archive/%f || cp %p /var/lib/postgresql/archive/%f'
primary_conninfo = 'user=student port=5432'
primary_slot_name = 'replica'
EOF
```

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

The replica also needs a `standby.signal` file.

```
postgres$ touch /var/lib/postgresql/16/beta/standby.signal
```

Start the primary and create a replication slot.

```
student$ sudo pg_ctlcluster 16 alpha start
```

```
student$ pg_receivewal --create-slot --slot=replica
```

Start the replica.

```
student$ sudo pg_ctlcluster 16 beta start
```

```
student$ psql -d replica_switchover
```

Repeat the experiment. Switch the segment:

```
α=> SELECT pg_switch_wal();
```

```
pg_switch_wal
-----
0/3006620
(1 row)
```

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

```
INSERT 0 1
```

Checking the archiving status:

```
α=> SELECT
```

```
  pg_walfile_name(pg_current_wal_lsn()) current_wal,
  last_archived_wal,
  last_failed_wal
FROM pg_stat_archiver;
```

```

current_wal | last_archived_wal | last_failed_wal
-----+-----+-----
000000010000000000000004 | 000000010000000000000003 |
(1 row)
```

The completed segment has been archived.

The alpha server is experiencing archiving issues – the command returns 1:

```
α=> ALTER SYSTEM SET archive_command = 'exit 1';
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 alpha reload
```

Alpha continues to generate WAL segments.

```
α=> SELECT pg_switch_wal();
```

```
pg_switch_wal
-----
0/4002BE0
(1 row)
```

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

```
INSERT 0 1
```

But the primary does not archive them:

```
α=> SELECT
```

```
  pg_walfile_name(pg_current_wal_lsn()) current_wal,
  last_archived_wal,
  last_failed_wal
FROM pg_stat_archiver;
```

```

current_wal | last_archived_wal | last_failed_wal
-----+-----+-----
000000010000000000000005 | 000000010000000000000003 | 000000010000000000000004
(1 row)
```

However, archiving from the replica succeeds and the segment ends up in the archive:

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

```
total 32768
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 000000010000000000000003
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 000000010000000000000004
```

Promote the replica.

```
student$ sudo kill -QUIT $(sudo head -n 1 /var/lib/postgresql/16/alpha/postmaster.pid)
```

```
student$ sudo pg_ctlcluster 16 beta promote
```

Beta is now the primary server and is generating WAL files.

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

```
| β=> SELECT pg_switch_wal();
```

```
pg_switch_wal
-----
0/5002190
(1 row)
```

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

```
INSERT 0 1
```

Let's check the archive again:

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

```
total 65540
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000001000000000000000003
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000001000000000000000004
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000001000000000000000005.partial
-rw----- 1 postgres postgres 16777216 Sep 17 00:01 00000002000000000000000005
-rw----- 1 postgres postgres      41 Sep 17 00:01 00000002.history
```

The replica's files have appeared in the archive, there are no gaps, and the problem is resolved.

Switchover is used both during normal operation and in emergency situations

After a switchover, the former leader must be brought back online

Both procedures must be tried and tested in advance

The write-ahead log archive requires extra attention

1. Set up streaming replication from alpha to beta, with a replication slot, without continuous archiving.
2. Simulate alpha server failure and switch over to beta.
3. Bring alpha back online as a replica by backing up from the new leader beta and adjusting the configuration.

Verify that replication is functioning and using the slot.

4. Switch back to alpha so that the former leader becomes the main server again.

To simulate a failure, you can stop the server in immediate mode:

```
pg_ctlcluster 16 cluster_name stop -m immediate --skip-systemctl-redirect
```

or terminate the main postgres process:

```
kill -QUIT номер_процесса
```

The process ID is the first line of the PGDATA/postmaster.pid file.

After completing the practice tasks, delete the slot and the created database, and reset the configuration.

1. Setting Up Replication Without an Archive

Set up replication.

Stop the beta server and clear its data directory.

```
student$ sudo pg_ctlcluster 16 beta stop
```

Cluster is not running.

```
postgres$ rm -rf /var/lib/postgresql/16/beta
```

Create a standalone backup in the beta server's data directory, first creating the replica slot.

```
postgres$ pg_basebackup \
--checkpoint=fast \
--pgdata=/var/lib/postgresql/16/beta \
-R \
--create-slot \
--slot=replica
```

```
postgres$ cat /var/lib/postgresql/16/beta/postgresql.auto.conf
```

```
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=postgres passfile='/var/lib/postgresql/.pgpass'
channel_binding=prefer host='/var/run/postgresql' port=5432 sslmode=prefer
sslnegotiation=postgres sslcompression=0 sslcertmode=allow sslsni=1
ssl_min_protocol_version=TLsv1.2 gssencmode=prefer krbsrvname=postgres gssdelegation=0
target_session_attrs=any load_balance_hosts=disable'
primary_slot_name = 'replica'
```

Start the server in replica mode.

```
student$ sudo pg_ctlcluster 16 beta start
```

Check the configuration. Run several commands on the primary server:

```
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 ('Hello, world!');
```

INSERT 0 1

Check the replica:

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

```
| β=> \c replica_switchover
```

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

```
| β=> SELECT * FROM test;
```

```
|      s
|-----
| Hello, world!
| (1 row)
```

2. Primary Server Failure and Failover to Replica

Simulate a primary server crash.

```
student$ sudo kill -QUIT $(sudo head -n 1 /var/lib/postgresql/16/alpha/postmaster.pid)
```

Promote the replica.

```
student$ sudo pg_ctlcluster 16 beta promote
```

3. Failback on Former Primary

Create a standalone backup of the beta server in the alpha server's data directory.

Before doing that, delete the configuration file – otherwise pg_basebackup will copy it and append parameters to it.

```
postgres$ rm -f /var/lib/postgresql/16/beta/postgresql.auto.conf
```

Clear the data directory on alpha.

```
postgres$ rm -rf /var/lib/postgresql/16/alpha
```

Run the backup utility to create a replication slot and replication configuration.

```
postgres$ pg_basebackup \
--port=5433 \
--checkpoint=fast \
--pgdata=/var/lib/postgresql/16/alpha \
-R \
--create-slot \
--slot=replica
```

Configuration parameters and the standby signal file have been set up:

```
postgres$ cat /var/lib/postgresql/16/alpha/postgresql.auto.conf
```

```
primary_conninfo = 'user=postgres passfile='/var/lib/postgresql/.pgpass'
channel_binding=prefer port=5433 sslmode=prefer sslnegotiation=postgres sslcompression=0
sslcertmode=allow sslsni=1 ssl_min_protocol_version=TLSv1.2 gssencmode=prefer
krbsrvname=postgres gssdelegation=0 target_session_attrs=any load_balance_hosts=disable'
primary_slot_name = 'replica'
```

```
postgres$ ls -l /var/lib/postgresql/16/alpha/standby.signal
```

```
-rw----- 1 postgres postgres 0 Sep 17 00:13 /var/lib/postgresql/16/alpha/standby.signal
```

```
student$ sudo pg_ctlcluster 16 alpha start
```

The replication slot has been initialized and is now in use:

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

```
| -[ RECORD 1 ]-----+-----
| slot_name      | replica
| plugin         |
| slot_type      | physical
| datoid         |
| database       |
| temporary     | f
| active         | t
| active_pid     | 105330
| xmin           |
| catalog_xmin   |
| restart_lsn    | 0/5000060
| confirmed_flush_lsn |
| wal_status     | reserved
| safe_wal_size  |
| two_phase     | f
| conflicting    |
```

Check the replication:

```
| β=> INSERT INTO test VALUES ('I am the former replica (now primary).');
```

```
| INSERT 0 1
```

```
student$ psql -d replica_switchover
```

```
α=> SELECT * FROM test;
```

```

      s
-----
Hello, world!
I am the former replica (now primary).
(2 rows)
```

4. Switching to a New Replica

Promote the replica.

```
student$ sudo pg_ctlcluster 16 alpha promote
```

```
α=> select pg_is_in_recovery();
```

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

The former primary is now primary again.

Cleanup

Delete the replication slot and database, and reset configuration.

```
α=> \c postgres
```

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

```
α=> DROP DATABASE replica_switchover;
```

```
DROP DATABASE
```

```
student$ sudo pg_ctlcluster 16 alpha restart
```

```
| β=> \c postgres
```

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

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

```
| pg_drop_replication_slot
| -----
|
| (1 row)
```

```
student$ sudo pg_ctlcluster 16 beta stop
```

1. Configure the alpha server to retain two additional WAL segments after each checkpoint. Start physical replication from alpha to beta without a replication slot.
2. Crash alpha and promote beta. Restart alpha and modify some data on it. Verify that the data on alpha and beta do not match.
3. Stop alpha and start it as a replica of beta using `pg_rewind`.

1. Set up replication with the simplest settings and without a replication slot. To ensure that two additional WAL segments are retained, set a suitable `wal_keep_size` value.

After completing the practice tasks, delete the slot and the created database, and reset the configuration.

1. Replication

Since we are not using replication slots, we will retain two additional WAL file segments.

```
α=> ALTER SYSTEM SET wal_keep_size = 32;
```

```
ALTER SYSTEM
```

Stop the beta server and make it a replica of the alpha server.

```
student$ sudo pg_ctlcluster 16 beta stop
```

Cluster is not running.

```
postgres$ rm -rf /var/lib/postgresql/16/beta
```

```
postgres$ pg_basebackup --checkpoint=fast -R --pgdata=/var/lib/postgresql/16/beta
```

```
student$ sudo pg_ctlcluster 16 beta start
```

Create a database and a table in it.

```
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 AS SELECT 'alpha is primary.' txt;
```

```
SELECT 1
```

Changes have been replicated to beta.

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

```
| β=> SELECT * FROM test;
```

```
|      txt      |
|-----|
| alpha is primary. |
| (1 row)        |
```

Check the replication status.

```
α=> SELECT * FROM pg_stat_replication \gx
```

```
-[ RECORD 1 ]-----+-----
pid          | 106207
usesysid     | 10
username     | postgres
application_name | 16/beta
client_addr  |
client_hostname |
client_port  | -1
backend_start | 2026-09-17 00:14:57.461258+03
backend_xmin  |
state        | streaming
sent_lsn     | 0/3432298
write_lsn    | 0/3432298
flush_lsn    | 0/3432298
replay_lsn   | 0/3432298
write_lag    | 00:00:00.000277
flush_lag    | 00:00:00.008669
replay_lag   | 00:00:00.008694
sync_priority | 0
sync_state   | async
reply_time   | 2026-09-17 00:15:04.970441+03
```

On the replica, you can view the WAL receiver statistics:

```
| β=> SELECT pid, status, written_lsn FROM pg_stat_wal_receiver;
```

```

    pid | status | written_lsn
-----+-----+-----
 106206 | streaming | 0/3432298
(1 row)

```

2. Replication Break

The primary server has lost power.

```
student$ sudo kill -QUIT $(sudo head -n 1 /var/lib/postgresql/16/alpha/postmaster.pid)
```

Promote beta.

```
student$ sudo pg_ctlcluster 16 beta promote
```

Make changes on the new primary.

```

β=> INSERT INTO test VALUES('beta promoted, handles write transactions.');
```

```

INSERT 0 1

```

Power has been restored to the alpha server.

```
student$ sudo pg_ctlcluster 16 alpha start
```

```
student$ psql -d replica_switchover
```

```

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

```

INSERT 0 1

```

The data on the two servers has diverged.

```
α=> SELECT * FROM test;
```

```

      txt
-----
alpha is primary.
Die Hard...
(2 rows)

```

```

β=> SELECT * FROM test;
```

```

      txt
-----
alpha is primary.
beta promoted, handles write transactions.
(2 rows)

```

Stop alpha (former primary).

```
student$ sudo pg_ctlcluster 16 alpha stop
```

3. Failback on Former Primary

We want to bring the alpha server back up as a replica. To do this, we need to bring it back to a state compatible with beta. We can use the stock utility for that.

```

postgres$ /usr/lib/postgresql/16/bin/pg_rewind \
-D /var/lib/postgresql/16/alpha \
--config-file=/etc/postgresql/16/alpha/postgresql.conf \
--source-server='user=postgres port=5433' \
-R

```

```

pg_rewind: servers diverged at WAL location 0/3432298 on timeline 1
pg_rewind: rewinding from last common checkpoint at 0/2000060 on timeline 1
pg_rewind: Done!

```

Write transactions are running on the beta server.

```

β=> INSERT INTO test VALUES('Changes still appear on beta.');
```

```

INSERT 0 1

```

Data changed on the beta server has been received by the alpha server:

```
student$ sudo pg_ctlcluster 16 alpha start
```

```
student$ psql -d replica_switchover
```

```
α=> SELECT * FROM test;
```

```
txt
```

```
-----  
alpha is primary.  
beta promoted, handles write transactions.  
Changes still appear on beta.  
(3 rows)
```

Cleanup

Delete the database we created and reset configuration.

```
| β=> \c postgres postgres
```

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

```
| β=> DROP DATABASE replica_switchover;
```

```
| DROP DATABASE
```

```
| β=> ALTER SYSTEM RESET ALL;
```

```
| ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 beta stop
```

```
α=> \c postgres postgres
```

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

```
α=> SELECT pg_promote();
```

```
pg_promote
```

```
-----  
t
```

```
(1 row)
```

```
α=> ALTER SYSTEM RESET ALL;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 alpha restart
```