NAME

pg_resetwal - reset the write-ahead log and other control information of a PostgreSQL database cluster

SYNOPSIS

pg_resetwal [**-f** | **--force**] [**-n** | **--dry-run**] [*option*...] [**-D** | **--pgdata**]*datadir*

DESCRIPTION

pg_resetwal clears the write-ahead log (WAL) and optionally resets some other control information stored in the pg_control file. This function is sometimes needed if these files have become corrupted. It should be used only as a last resort, when the server will not start due to such corruption.

After running this command, it should be possible to start the server, but bear in mind that the database might contain inconsistent data due to partially-committed transactions. You should immediately dump your data, run **initdb**, and restore. After restore, check for inconsistencies and repair as needed.

This utility can only be run by the user who installed the server, because it requires read/write access to the data directory. For safety reasons, you must specify the data directory on the command line. **pg_resetwal** does not use the environment variable **PGDATA**.

If **pg_resetwal** complains that it cannot determine valid data for pg_control, you can force it to proceed anyway by specifying the **-f** (force) option. In this case plausible values will be substituted for the missing data. Most of the fields can be expected to match, but manual assistance might be needed for the next OID, next transaction ID and epoch, next multitransaction ID and offset, and WAL starting location fields. These fields can be set using the options discussed below. If you are not able to determine correct values for all these fields, **-f** can still be used, but the recovered database must be treated with even more suspicion than usual: an immediate dump and restore is imperative. *Do not* execute any data-modifying operations in the database before you dump, as any such action is likely to make the corruption worse.

OPTIONS

-f

--force

Force **pg_resetwal** to proceed even if it cannot determine valid data for pg_control, as explained above.

-n

--dry-run

The **-n**/**--dry-run** option instructs **pg_resetwal** to print the values reconstructed from pg_control and values about to be changed, and then exit without modifying anything. This is mainly a debugging tool, but can be useful as a sanity check before allowing **pg_resetwal** to proceed for

real.

-V
--version
Display version information, then exit.

-?
--help
Show help, then exit.

The following options are only needed when **pg_resetwal** is unable to determine appropriate values by reading pg_control. Safe values can be determined as described below. For values that take numeric arguments, hexadecimal values can be specified by using the prefix 0x.

-c *xid*,*xid*
--commit-timestamp-ids=*xid*,*xid*
Manually set the oldest and newest transaction IDs for which the commit time can be retrieved.

A safe value for the oldest transaction ID for which the commit time can be retrieved (first part) can be determined by looking for the numerically smallest file name in the directory pg_commit_ts under the data directory. Conversely, a safe value for the newest transaction ID for which the commit time can be retrieved (second part) can be determined by looking for the numerically greatest file name in the same directory. The file names are in hexadecimal.

-e *xid_epoch*
--epoch=*xid_epoch*
Manually set the next transaction ID's epoch.

The transaction ID epoch is not actually stored anywhere in the database except in the field that is set by **pg_resetwal**, so any value will work so far as the database itself is concerned. You might need to adjust this value to ensure that replication systems such as Slony-I and Skytools work correctly -- if so, an appropriate value should be obtainable from the state of the downstream replicated database.

-l *walfile*
--next-wal-file=*walfile*
Manually set the WAL starting location by specifying the name of the next WAL segment file.

The name of next WAL segment file should be larger than any WAL segment file name currently existing in the directory pg_wal under the data directory. These names are also in hexadecimal

and have three parts. The first part is the "timeline ID" and should usually be kept the same. For example, if 000000010000000320000004A is the largest entry in pg_wal, use -l 000000010000000320000004B or higher.

Note that when using nondefault WAL segment sizes, the numbers in the WAL file names are different from the LSNs that are reported by system functions and system views. This option takes a WAL file name, not an LSN.

Note
pg_resetwal itself looks at the files in pg_wal and chooses a default **-l** setting beyond the last existing file name. Therefore, manual adjustment of **-l** should only be needed if you are aware of WAL segment files that are not currently present in pg_wal, such as entries in an offline archive; or if the contents of pg_wal have been lost entirely.

-m *mxid***,***mxid*
--multixact-ids=*mxid***,***mxid*
Manually set the next and oldest multitransaction ID.

A safe value for the next multitransaction ID (first part) can be determined by looking for the numerically largest file name in the directory pg_multixact/offsets under the data directory, adding one, and then multiplying by 65536 (0x10000). Conversely, a safe value for the oldest multitransaction ID (second part of **-m**) can be determined by looking for the numerically smallest file name in the same directory and multiplying by 65536. The file names are in hexadecimal, so the easiest way to do this is to specify the option value in hexadecimal and append four zeroes.

-o *oid*
--next-oid=*oid*
Manually set the next OID.

There is no comparably easy way to determine a next OID that’s beyond the largest one in the database, but fortunately it is not critical to get the next-OID setting right.

-O *mxoff*
--multixact-offset=*mxoff*
Manually set the next multitransaction offset.

A safe value can be determined by looking for the numerically largest file name in the directory pg_multixact/members under the data directory, adding one, and then multiplying by 52352 (0xCC80). The file names are in hexadecimal. There is no simple recipe such as the ones for other options of appending zeroes.

--wal-segsize=*wal_segment_size*

Set the new WAL segment size, in megabytes. The value must be set to a power of 2 between 1 and 1024 (megabytes). See the same option of **initdb**(1) for more information.

Note

While **pg_resetwal** will set the WAL starting address beyond the latest existing WAL segment file, some segment size changes can cause previous WAL file names to be reused. It is recommended to use **-l** together with this option to manually set the WAL starting address if WAL file name overlap will cause problems with your archiving strategy.

-u *xid*

--oldest-transaction-id=*xid*

Manually set the oldest unfrozen transaction ID.

A safe value can be determined by looking for the numerically smallest file name in the directory pg_xact under the data directory and then multiplying by 1048576 (0x100000). Note that the file names are in hexadecimal. It is usually easiest to specify the option value in hexadecimal too. For example, if 0007 is the smallest entry in pg_xact, -u 0x700000 will work (five trailing zeroes provide the proper multiplier).

-x *xid*

--next-transaction-id=*xid*

Manually set the next transaction ID.

A safe value can be determined by looking for the numerically largest file name in the directory pg_xact under the data directory, adding one, and then multiplying by 1048576 (0x100000). Note that the file names are in hexadecimal. It is usually easiest to specify the option value in hexadecimal too. For example, if 0011 is the largest entry in pg_xact, -x 0x1200000 will work (five trailing zeroes provide the proper multiplier).

ENVIRONMENT

PG_COLOR

Specifies whether to use color in diagnostic messages. Possible values are always, auto and never.

NOTES

This command must not be used when the server is running. **pg_resetwal** will refuse to start up if it finds a server lock file in the data directory. If the server crashed then a lock file might have been left behind; in that case you can remove the lock file to allow **pg_resetwal** to run. But before you do so, make doubly certain that there is no server process still alive.

pg_resetwal works only with servers of the same major version.

SEE ALSO

pg_controldata(1)