NAME

pg_rewind - synchronize a PostgreSQL data directory with another data directory that was forked from it

SYNOPSIS

pg_rewind [*option*...] {**-D** | **--target-pgdata**} *directory* {**--source-pgdata=***directory* | **--source-server=***connstr*}

DESCRIPTION

pg_rewind is a tool for synchronizing a PostgreSQL cluster with another copy of the same cluster, after the clusters' timelines have diverged. A typical scenario is to bring an old primary server back online after failover as a standby that follows the new primary.

After a successful rewind, the state of the target data directory is analogous to a base backup of the source data directory. Unlike taking a new base backup or using a tool like rsync, pg_rewind does not require comparing or copying unchanged relation blocks in the cluster. Only changed blocks from existing relation files are copied; all other files, including new relation files, configuration files, and WAL segments, are copied in full. As such the rewind operation is significantly faster than other approaches when the database is large and only a small fraction of blocks differ between the clusters.

pg_rewind examines the timeline histories of the source and target clusters to determine the point where they diverged, and expects to find WAL in the target cluster's pg_wal directory reaching all the way back to the point of divergence. The point of divergence can be found either on the target timeline, the source timeline, or their common ancestor. In the typical failover scenario where the target cluster was shut down soon after the divergence, this is not a problem, but if the target cluster ran for a long time after the divergence, its old WAL files might no longer be present. In this case, you can manually copy them from the WAL archive to the pg_wal directory, or run pg_rewind with the -c option to automatically retrieve them from the WAL archive. The use of pg_rewind is not limited to failover, e.g., a standby server can be promoted, run some write transactions, and then rewound to become a standby again.

After running pg_rewind, WAL replay needs to complete for the data directory to be in a consistent state. When the target server is started again it will enter archive recovery and replay all WAL generated in the source server from the last checkpoint before the point of divergence. If some of the WAL was no longer available in the source server when pg_rewind was run, and therefore could not be copied by the pg_rewind session, it must be made available when the target server is started. This can be done by creating a recovery.signal file in the target data directory and by configuring a suitable restore_command in postgresql.conf.

pg_rewind requires that the target server either has the wal_log_hints option enabled in postgresql.conf or data checksums enabled when the cluster was initialized with initdb. Neither of these are currently on by default. full_page_writes must also be set to on, but is enabled by default.

> **Warning**
>
> If pg_rewind fails while processing, then the data folder of the target is likely not in a state that can be recovered. In such a case, taking a new fresh backup is recommended.
>
> As pg_rewind copies configuration files entirely from the source, it may be required to correct the configuration used for recovery before restarting the target server, especially if the target is reintroduced as a standby of the source. If you restart the server after the rewind operation has finished but without configuring recovery, the target may again diverge from the primary.
>
> pg_rewind will fail immediately if it finds files it cannot write directly to. This can happen for example when the source and the target server use the same file mapping for read-only SSL keys and certificates. If such files are present on the target server it is recommended to remove them before running pg_rewind. After doing the rewind, some of those files may have been copied from the source, in which case it may be necessary to remove the data copied and restore back the set of links used before the rewind.

OPTIONS

pg_rewind accepts the following command-line arguments:

-D *directory*
--target-pgdata=*directory*
: This option specifies the target data directory that is synchronized with the source. The target server must be shut down cleanly before running pg_rewind

--source-pgdata=*directory*
: Specifies the file system path to the data directory of the source server to synchronize the target with. This option requires the source server to be cleanly shut down.

--source-server=*connstr*
: Specifies a libpq connection string to connect to the source PostgreSQL server to synchronize the target with. The connection must be a normal (non-replication) connection with a role having sufficient permissions to execute the functions used by pg_rewind on the source server (see Notes section for details) or a superuser role. This option requires the source server to be running and accepting connections.

-R

--write-recovery-conf
Create standby.signal and append connection settings to postgresql.auto.conf in the output directory. --source-server is mandatory with this option.

-n
--dry-run
Do everything except actually modifying the target directory.

-N
--no-sync
By default, **pg_rewind** will wait for all files to be written safely to disk. This option causes **pg_rewind** to return without waiting, which is faster, but means that a subsequent operating system crash can leave the data directory corrupt. Generally, this option is useful for testing but should not be used on a production installation.

-P
--progress
Enables progress reporting. Turning this on will deliver an approximate progress report while copying data from the source cluster.

-c
--restore-target-wal
Use *restore_command* defined in the target cluster configuration to retrieve WAL files from the WAL archive if these files are no longer available in the pg_wal directory.

--config-file=*filename*
Use the specified main server configuration file for the target cluster. This affects pg_rewind when it uses internally the postgres command for the rewind operation on this cluster (when retrieving *restore_command* with the option **-c/--restore-target-wal** and when forcing a completion of crash recovery).

--debug
Print verbose debugging output that is mostly useful for developers debugging pg_rewind.

--no-ensure-shutdown
pg_rewind requires that the target server is cleanly shut down before rewinding. By default, if the target server is not shut down cleanly, pg_rewind starts the target server in single-user mode to complete crash recovery first, and stops it. By passing this option, pg_rewind skips this and errors out immediately if the server is not cleanly shut down. Users are expected to handle the situation themselves in that case.

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

-?
--help
Show help, then exit.

ENVIRONMENT

When **--source-server** option is used, pg_rewind also uses the environment variables supported by libpq (see Section 34.15).

The environment variable **PG_COLOR** specifies whether to use color in diagnostic messages. Possible values are always, auto and never.

NOTES

When executing pg_rewind using an online cluster as source, a role having sufficient permissions to execute the functions used by pg_rewind on the source cluster can be used instead of a superuser. Here is how to create such a role, named rewind_user here:

```
CREATE USER rewind_user LOGIN;
GRANT EXECUTE ON function pg_catalog.pg_ls_dir(text, boolean, boolean) TO rewind_user;
GRANT EXECUTE ON function pg_catalog.pg_stat_file(text, boolean) TO rewind_user;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text) TO rewind_user;
GRANT EXECUTE ON function pg_catalog.pg_read_binary_file(text, bigint, bigint, boolean) TO rewind_user;
```

When executing pg_rewind using an online cluster as source which has been recently promoted, it is necessary to execute a **CHECKPOINT** after promotion such that its control file reflects up-to-date timeline information, which is used by pg_rewind to check if the target cluster can be rewound using the designated source cluster.

How It Works

The basic idea is to copy all file system-level changes from the source cluster to the target cluster:

1.
the WAL log of the target cluster, starting from the last checkpoint before the point where the source cluster's timeline history forked off from the target cluster. For each WAL record, record each data block that was touched. This yields a list of all the data blocks that were changed in the target cluster, after the source cluster forked off. If some of the WAL files are no longer available, try re-running pg_rewind with the **-c** option to search for the missing files in the WAL archive.

2.
all those changed blocks from the source cluster to the target cluster, either using direct file system access (**--source-pgdata**) or SQL (**--source-server**). Relation files are now in a state equivalent to the moment of the last completed checkpoint prior to the point at which the WAL timelines of the source and target diverged plus the current state on the source of any blocks changed on the target after that divergence.

3.
all other files, including new relation files, WAL segments, pg_xact, and configuration files from the source cluster to the target cluster. Similarly to base backups, the contents of the directories pg_dynshmem/, pg_notify/, pg_replslot/, pg_serial/, pg_snapshots/, pg_stat_tmp/, and pg_subtrans/ are omitted from the data copied from the source cluster. The files backup_label, tablespace_map, pg_internal.init, postmaster.opts, and postmaster.pid, as well as any file or directory beginning with pgsql_tmp, are omitted.

4.
a backup_label file to begin WAL replay at the checkpoint created at failover and configure the pg_control file with a minimum consistency LSN defined as the result of pg_current_wal_insert_lsn() when rewinding from a live source or the last checkpoint LSN when rewinding from a stopped source.

5.
starting the target, PostgreSQL replays all the required WAL, resulting in a data directory in a consistent state.