NAME

pg_basebackup - take a base backup of a PostgreSQL cluster

SYNOPSIS

pg_basebackup [*option*...]

DESCRIPTION

pg_basebackup is used to take a base backup of a running PostgreSQL database cluster. The backup is taken without affecting other clients of the database, and can be used both for point-in-time recovery (see Section 26.3) and as the starting point for a log-shipping or streaming-replication standby server (see Section 27.2).

pg_basebackup makes an exact copy of the database cluster's files, while making sure the server is put into and out of backup mode automatically. Backups are always taken of the entire database cluster; it is not possible to back up individual databases or database objects. For selective backups, another tool such as **pg_dump**(1) must be used.

The backup is made over a regular PostgreSQL connection that uses the replication protocol. The connection must be made with a user ID that has REPLICATION permissions (see Section 22.2) or is a superuser, and pg_hba.conf must permit the replication connection. The server must also be configured with max_wal_senders set high enough to provide at least one walsender for the backup plus one for WAL streaming (if used).

There can be multiple **pg_basebackup**s running at the same time, but it is usually better from a performance point of view to take only one backup, and copy the result.

pg_basebackup can make a base backup from not only a primary server but also a standby. To take a backup from a standby, set up the standby so that it can accept replication connections (that is, set *max_wal_senders* and hot_standby, and configure its pg_hba.conf appropriately). You will also need to enable full_page_writes on the primary.

Note that there are some limitations in taking a backup from a standby:

- backup history file is not created in the database cluster backed up.

- cannot force the standby to switch to a new WAL file at the end of backup. When you are using -X none, if write activity on the primary is low, pg_basebackup may need to wait a long time for the last WAL file required for the backup to be switched and archived. In this case, it may be useful to run **pg_switch_wal**

on the primary in order to trigger an immediate WAL file switch.

- the standby is promoted to be primary during backup, the backup fails.

- WAL records required for the backup must contain sufficient full-page writes, which requires you to enable *full_page_writes* on the primary.

Whenever pg_basebackup is taking a base backup, the server's pg_stat_progress_basebackup view will report the progress of the backup. See Section 28.4.5 for details.

OPTIONS

The following command-line options control the location and format of the output:

-D *directory*
--pgdata=*directory*

Sets the target directory to write the output to. pg_basebackup will create this directory (and any missing parent directories) if it does not exist. If it already exists, it must be empty.

When the backup is in tar format, the target directory may be specified as - (dash), causing the tar file to be written to stdout.

This option is required.

-F *format*
--format=*format*

Selects the format for the output. *format* can be one of the following:

p
plain

Write the output as plain files, with the same layout as the source server's data directory and tablespaces. When the cluster has no additional tablespaces, the whole database will be placed in the target directory. If the cluster contains additional tablespaces, the main data directory will be placed in the target directory, but all other tablespaces will be placed in the same absolute path as they have on the source server. (See **--tablespace-mapping** to change that.)

This is the default format.

t
tar
Write the output as tar files in the target directory. The main data directory's contents will be written to a file named base.tar, and each other tablespace will be written to a separate tar file named after that tablespace's OID.

If the target directory is specified as - (dash), the tar contents will be written to standard output, suitable for piping to (for example) gzip. This is only allowed if the cluster has no additional tablespaces and WAL streaming is not used.

-R
--write-recovery-conf
Creates a standby.signal

file and appends connection settings to the postgresql.auto.conf file in the target directory (or within the base archive file when using tar format). This eases setting up a standby server using the results of the backup.

The postgresql.auto.conf file will record the connection settings and, if specified, the replication slot that pg_basebackup is using, so that streaming replication will use the same settings later on.

-t *target*
--target=*target*
Instructs the server where to place the base backup. The default target is client, which specifies that the backup should be sent to the machine where pg_basebackup is running. If the target is instead set to server:/some/path, the backup will be stored on the machine where the server is running in the /some/path directory. Storing a backup on the server requires superuser privileges or having privileges of the pg_write_server_files role. If the target is set to blackhole, the contents are discarded and not stored anywhere. This should only be used for testing purposes, as you will not end up with an actual backup.

Since WAL streaming is implemented by pg_basebackup rather than by the server, this option cannot be used together with -Xstream. Since that is the default, when this option is specified, you must also specify either -Xfetch or -Xnone.

-T *olddir=newdir*
--tablespace-mapping=*olddir=newdir*
Relocates the tablespace in directory *olddir* to *newdir* during the backup. To be effective, *olddir* must exactly match the path specification of the tablespace as it is defined on the source server. (But it is not an error if there is no tablespace in *olddir* on the source server.) Meanwhile *newdir* is

a directory in the receiving host's filesystem. As with the main target directory, *newdir* need not exist already, but if it does exist it must be empty. Both *olddir* and *newdir* must be absolute paths. If either path needs to contain an equal sign (=), precede that with a backslash. This option can be specified multiple times for multiple tablespaces.

If a tablespace is relocated in this way, the symbolic links inside the main data directory are updated to point to the new location. So the new data directory is ready to be used for a new server instance with all tablespaces in the updated locations.

Currently, this option only works with plain output format; it is ignored if tar format is selected.

--waldir=*waldir*

Sets the directory to write WAL (write-ahead log) files to. By default WAL files will be placed in the pg_wal subdirectory of the target directory, but this option can be used to place them elsewhere. *waldir* must be an absolute path. As with the main target directory, *waldir* need not exist already, but if it does exist it must be empty. This option can only be specified when the backup is in plain format.

-X *method*

--wal-method=*method*

Includes the required WAL (write-ahead log) files in the backup. This will include all write-ahead logs generated during the backup. Unless the method none is specified, it is possible to start a postmaster in the target directory without the need to consult the log archive, thus making the output a completely standalone backup.

The following *method*s for collecting the write-ahead logs are supported:

n
none
: Don't include write-ahead logs in the backup.

f
fetch
: The write-ahead log files are collected at the end of the backup. Therefore, it is necessary for the source server's wal_keep_size parameter to be set high enough that the required log data is not removed before the end of the backup. If the required log data has been recycled before it's time to transfer it, the backup will fail and be unusable.

When tar format is used, the write-ahead log files will be included in the base.tar file.

s

stream

Stream write-ahead log data while the backup is being taken. This method will open a second connection to the server and start streaming the write-ahead log in parallel while running the backup. Therefore, it will require two replication connections not just one. As long as the client can keep up with the write-ahead log data, using this method requires no extra write-ahead logs to be saved on the source server.

When tar format is used, the write-ahead log files will be written to a separate file named pg_wal.tar (if the server is a version earlier than 10, the file will be named pg_xlog.tar).

This value is the default.

-z

--gzip

Enables gzip compression of tar file output, with the default compression level. Compression is only available when using the tar format, and the suffix .gz will automatically be added to all tar filenames.

-Z *level*

-Z [{client|server}-]*method***[:***detail***]**

--compress=*level*

--compress=[{client|server}-]*method***[:***detail***]**

Requests compression of the backup. If client or server is included, it specifies where the compression is to be performed. Compressing on the server will reduce transfer bandwidth but will increase server CPU consumption. The default is client except when --target is used. In that case, the backup is not being sent to the client, so only server compression is sensible. When -Xstream, which is the default, is used, server-side compression will not be applied to the WAL. To compress the WAL, use client-side compression, or specify -Xfetch.

The compression method can be set to gzip, lz4, zstd, or none for no compression. A compression detail string can optionally be specified. If the detail string is an integer, it specifies the compression level. Otherwise, it should be a comma-separated list of items, each of the form keyword or keyword=value. Currently, the supported keywords are level and workers.

If no compression level is specified, the default compression level will be used. If only a level is specified without mentioning an algorithm, gzip compression will be used if the level is greater than 0, and no compression will be used if the level is 0.

When the tar format is used with gzip, lz4, or zstd, the suffix .gz, .lz4, or .zst, respectively, will be

automatically added to all tar filenames. When the plain format is used, client-side compression may not be specified, but it is still possible to request server-side compression. If this is done, the server will compress the backup for transmission, and the client will decompress and extract it.

When this option is used in combination with -Xstream, pg_wal.tar will be compressed using gzip if client-side gzip compression is selected, but will not be compressed if any other compression algorithm is selected, or if server-side compression is selected.

The following command-line options control the generation of the backup and the invocation of the program:

-c {fast|spread}
--checkpoint={fast|spread}
Sets checkpoint mode to fast (immediate) or spread (the default) (see Section 26.3.3).

-C
--create-slot
Specifies that the replication slot named by the --slot option should be created before starting the backup. An error is raised if the slot already exists.

-l *label*
--label=*label*
Sets the label for the backup. If none is specified, a default value of "pg_basebackup base backup" will be used.

-n
--no-clean
By default, when **pg_basebackup** aborts with an error, it removes any directories it might have created before discovering that it cannot finish the job (for example, the target directory and write-ahead log directory). This option inhibits tidying-up and is thus useful for debugging.

Note that tablespace directories are not cleaned up either way.

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

-P
--progress
Enables progress reporting. Turning this on will deliver an approximate progress report during the backup. Since the database may change during the backup, this is only an approximation and may not end at exactly 100%. In particular, when WAL log is included in the backup, the total amount of data cannot be estimated in advance, and in this case the estimated target size will increase once it passes the total estimate without WAL.

-r *rate*
--max-rate=*rate*
Sets the maximum transfer rate at which data is collected from the source server. This can be useful to limit the impact of pg_basebackup on the server. Values are in kilobytes per second. Use a suffix of M to indicate megabytes per second. A suffix of k is also accepted, and has no effect. Valid values are between 32 kilobytes per second and 1024 megabytes per second.

This option always affects transfer of the data directory. Transfer of WAL files is only affected if the collection method is fetch.

-S *slotname*
--slot=*slotname*
This option can only be used together with -X stream. It causes WAL streaming to use the specified replication slot. If the base backup is intended to be used as a streaming-replication standby using a replication slot, the standby should then use the same replication slot name as primary_slot_name. This ensures that the primary server does not remove any necessary WAL data in the time between the end of the base backup and the start of streaming replication on the new standby.

The specified replication slot has to exist unless the option **-C** is also used.

If this option is not specified and the server supports temporary replication slots (version 10 and later), then a temporary replication slot is automatically used for WAL streaming.

-v
--verbose
Enables verbose mode. Will output some extra steps during startup and shutdown, as well as show the exact file name that is currently being processed if progress reporting is also enabled.

--manifest-checksums=*algorithm*
Specifies the checksum algorithm that should be applied to each file included in the backup manifest. Currently, the available algorithms are NONE, CRC32C, SHA224, SHA256, SHA384,

and SHA512. The default is CRC32C.

If NONE is selected, the backup manifest will not contain any checksums. Otherwise, it will contain a checksum of each file in the backup using the specified algorithm. In addition, the manifest will always contain a SHA256 checksum of its own contents. The SHA algorithms are significantly more CPU-intensive than CRC32C, so selecting one of them may increase the time required to complete the backup.

Using a SHA hash function provides a cryptographically secure digest of each file for users who wish to verify that the backup has not been tampered with, while the CRC32C algorithm provides a checksum that is much faster to calculate; it is good at catching errors due to accidental changes but is not resistant to malicious modifications. Note that, to be useful against an adversary who has access to the backup, the backup manifest would need to be stored securely elsewhere or otherwise verified not to have been modified since the backup was taken.

pg_verifybackup(1) can be used to check the integrity of a backup against the backup manifest.

--manifest-force-encode
Forces all filenames in the backup manifest to be hex-encoded. If this option is not specified, only non-UTF8 filenames are hex-encoded. This option is mostly intended to test that tools which read a backup manifest file properly handle this case.

--no-estimate-size
Prevents the server from estimating the total amount of backup data that will be streamed, resulting in the backup_total column in the pg_stat_progress_basebackup view always being NULL.

Without this option, the backup will start by enumerating the size of the entire database, and then go back and send the actual contents. This may make the backup take slightly longer, and in particular it will take longer before the first data is sent. This option is useful to avoid such estimation time if it’s too long.

This option is not allowed when using **--progress**.

--no-manifest
Disables generation of a backup manifest. If this option is not specified, the server will generate and send a backup manifest which can be verified using **pg_verifybackup**(1). The manifest is a list of every file present in the backup with the exception of any WAL files that may be included. It also stores the size, last modification time, and an optional checksum for each file.

--no-slot
Prevents the creation of a temporary replication slot for the backup.

By default, if log streaming is selected but no slot name is given with the **-S** option, then a temporary replication slot is created (if supported by the source server).

The main purpose of this option is to allow taking a base backup when the server has no free replication slots. Using a replication slot is almost always preferred, because it prevents needed WAL from being removed by the server during the backup.

--no-verify-checksums
Disables verification of checksums, if they are enabled on the server the base backup is taken from.

By default, checksums are verified and checksum failures will result in a non-zero exit status. However, the base backup will not be removed in such a case, as if the **--no-clean** option had been used. Checksum verification failures will also be reported in the pg_stat_database view.

The following command-line options control the connection to the source server:

-d *connstr*
--dbname=*connstr*
Specifies parameters used to connect to the server, as a connection string; these will override any conflicting command line options.

The option is called --dbname for consistency with other client applications, but because pg_basebackup doesn't connect to any particular database in the cluster, any database name in the connection string will be ignored.

-h *host*
--host=*host*
Specifies the host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for a Unix domain socket. The default is taken from the **PGHOST** environment variable, if set, else a Unix domain socket connection is attempted.

-p *port*
--port=*port*
Specifies the TCP port or local Unix domain socket file extension on which the server is listening for connections. Defaults to the **PGPORT** environment variable, if set, or a compiled-in default.

-s *interval*
--status-interval=*interval*
Specifies the number of seconds between status packets sent back to the source server. Smaller values allow more accurate monitoring of backup progress from the server. A value of zero disables periodic status updates completely, although an update will still be sent when requested by the server, to avoid timeout-based disconnects. The default value is 10 seconds.

-U *username*
--username=*username*
Specifies the user name to connect as.

-w
--no-password
Prevents issuing a password prompt. If the server requires password authentication and a password is not available by other means such as a .pgpass file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

-W
--password
Forces pg_basebackup to prompt for a password before connecting to the source server.

This option is never essential, since pg_basebackup will automatically prompt for a password if the server demands password authentication. However, pg_basebackup will waste a connection attempt finding out that the server wants a password. In some cases it is worth typing **-W** to avoid the extra connection attempt.

Other options are also available:

-V
--version
Prints the pg_basebackup version and exits.

-?
--help
Shows help about pg_basebackup command line arguments, and exits.

ENVIRONMENT

This utility, like most other PostgreSQL utilities, 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

At the beginning of the backup, a checkpoint needs to be performed on the source server. This can take some time (especially if the option --checkpoint=fast is not used), during which pg_basebackup will appear to be idle.

The backup will include all files in the data directory and tablespaces, including the configuration files and any additional files placed in the directory by third parties, except certain temporary files managed by PostgreSQL. But only regular files and directories are copied, except that symbolic links used for tablespaces are preserved. Symbolic links pointing to certain directories known to PostgreSQL are copied as empty directories. Other symbolic links and special device files are skipped. See Section 55.4 for the precise details.

In plain format, tablespaces will be backed up to the same path they have on the source server, unless the option --tablespace-mapping is used. Without this option, running a plain format base backup on the same host as the server will not work if tablespaces are in use, because the backup would have to be written to the same directory locations as the original tablespaces.

When tar format is used, it is the user's responsibility to unpack each tar file before starting a PostgreSQL server that uses the data. If there are additional tablespaces, the tar files for them need to be unpacked in the correct locations. In this case the symbolic links for those tablespaces will be created by the server according to the contents of the tablespace_map file that is included in the base.tar file.

pg_basebackup works with servers of the same or an older major version, down to 9.1. However, WAL streaming mode (-X stream) only works with server version 9.3 and later, and tar format (--format=tar) only works with server version 9.5 and later.

pg_basebackup will preserve group permissions for data files if group permissions are enabled on the source cluster.

EXAMPLES

To create a base backup of the server at mydbserver and store it in the local directory /usr/local/pgsql/data:

```
$ pg_basebackup -h mydbserver -D /usr/local/pgsql/data
```

To create a backup of the local server with one compressed tar file for each tablespace, and store it in

the directory backup, showing a progress report while running:

```
$ pg_basebackup -D backup -Ft -z -P
```

To create a backup of a single-tablespace local database and compress this with bzip2:

```
$ pg_basebackup -D - -Ft -X fetch | bzip2 > backup.tar.bz2
```

(This command will fail if there are multiple tablespaces in the database.)

To create a backup of a local database where the tablespace in /opt/ts is relocated to ./backup/ts:

```
$ pg_basebackup -D backup/data -T /opt/ts=$(pwd)/backup/ts
```

To create a backup of a local server with one tar file for each tablespace compressed with gzip at level 9, stored in the directory backup:

```
$ pg_basebackup -D backup -Ft --compress=gzip:9
```

SEE ALSO

pg_dump(1), Section 28.4.5