NAME

pg_upgrade - upgrade a PostgreSQL server instance

SYNOPSIS

pg_upgrade -b *oldbindir* [**-B** *newbindir*] **-d** *oldconfigdir* **-D** *newconfigdir* [*option*...]

DESCRIPTION

pg_upgrade (formerly called pg_migrator) allows data stored in PostgreSQL data files to be upgraded to a later PostgreSQL major version without the data dump/restore typically required for major version upgrades, e.g., from 9.5.8 to 9.6.4 or from 10.7 to 11.2. It is not required for minor version upgrades, e.g., from 9.6.2 to 9.6.3 or from 10.1 to 10.2.

Major PostgreSQL releases regularly add new features that often change the layout of the system tables, but the internal data storage format rarely changes. pg_upgrade uses this fact to perform rapid upgrades by creating new system tables and simply reusing the old user data files. If a future major release ever changes the data storage format in a way that makes the old data format unreadable, pg_upgrade will not be usable for such upgrades. (The community will attempt to avoid such situations.)

pg_upgrade does its best to make sure the old and new clusters are binary-compatible, e.g., by checking for compatible compile-time settings, including 32/64-bit binaries. It is important that any external modules are also binary compatible, though this cannot be checked by pg_upgrade.

pg_upgrade supports upgrades from 9.2.X and later to the current major release of PostgreSQL, including snapshot and beta releases.

OPTIONS

pg_upgrade accepts the following command-line arguments:

-b *bindir*
--old-bindir=*bindir*
the old PostgreSQL executable directory; environment variable **PGBINOLD**

-B *bindir*
--new-bindir=*bindir*
the new PostgreSQL executable directory; default is the directory where pg_upgrade resides; environment variable **PGBINNEW**

-c
--check

check clusters only, don’t change any data

-d *configdir*
--old-datadir=*configdir*
the old database cluster configuration directory; environment variable **PGDATAOLD**

-D *configdir*
--new-datadir=*configdir*
the new database cluster configuration directory; environment variable **PGDATANEW**

-j *njobs*
--jobs=*njobs*
number of simultaneous processes or threads to use

-k
--link
use hard links instead of copying files to the new cluster

-N
--no-sync
By default, **pg_upgrade** will wait for all files of the upgraded cluster to be written safely to disk. This option causes **pg_upgrade** 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.

-o *options*
--old-options *options*
options to be passed directly to the old **postgres** command; multiple option invocations are appended

-O *options*
--new-options *options*
options to be passed directly to the new **postgres** command; multiple option invocations are appended

-p *port*
--old-port=*port*
the old cluster port number; environment variable **PGPORTOLD**

-P *port*

--new-port=*port*
the new cluster port number; environment variable **PGPORTNEW**

-r
--retain
retain SQL and log files even after successful completion

-s *dir*
--socketdir=*dir*
directory to use for postmaster sockets during upgrade; default is current working directory; environment variable **PGSOCKETDIR**

-U *username*
--username=*username*
cluster's install user name; environment variable **PGUSER**

-v
--verbose
enable verbose internal logging

-V
--version
display version information, then exit

--clone
Use efficient file cloning (also known as "reflinks" on some systems) instead of copying files to the new cluster. This can result in near-instantaneous copying of the data files, giving the speed advantages of **-k**/**--link** while leaving the old cluster untouched.

File cloning is only supported on some operating systems and file systems. If it is selected but not supported, the pg_upgrade run will error. At present, it is supported on Linux (kernel 4.5 or later) with Btrfs and XFS (on file systems created with reflink support), and on macOS with APFS.

-?
--help
show help, then exit

USAGE

These are the steps to perform an upgrade with pg_upgrade:

1.
move the old cluster: If you are using a version-specific installation directory, e.g., /opt/PostgreSQL/15, you do not need to move the old cluster. The graphical installers all use version-specific installation directories.

If your installation directory is not version-specific, e.g., /usr/local/pgsql, it is necessary to move the current PostgreSQL install directory so it does not interfere with the new PostgreSQL installation. Once the current PostgreSQL server is shut down, it is safe to rename the PostgreSQL installation directory; assuming the old directory is /usr/local/pgsql, you can do:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

to rename the directory.

2.
source installs, build the new version: Build the new PostgreSQL source with **configure** flags that are compatible with the old cluster. pg_upgrade will check **pg_controldata** to make sure all settings are compatible before starting the upgrade.

3.
the new PostgreSQL binaries: Install the new server's binaries and support files. pg_upgrade is included in a default installation.

For source installs, if you wish to install the new server in a custom location, use the prefix variable:

```
make prefix=/usr/local/pgsql.new install
```

4.
the new PostgreSQL cluster: Initialize the new cluster using **initdb**. Again, use compatible **initdb** flags that match the old cluster. Many prebuilt installers do this step automatically. There is no need to start the new cluster.

5.
extension shared object files: Many extensions and custom modules, whether from contrib or another source, use shared object files (or DLLs), e.g., pgcrypto.so. If the old cluster used these, shared object files matching the new server binary must be installed in the new cluster, usually via operating system commands. Do not load the schema definitions, e.g., **CREATE EXTENSION pgcrypto**, because these will be duplicated from the old cluster. If extension updates are available, pg_upgrade will report this and create a script that can be run later to update them.

6.
custom full-text search files: Copy any custom full text search files (dictionary, synonym, thesaurus, stop words) from the old to the new cluster.

7.
authentication: **pg_upgrade** will connect to the old and new servers several times, so you might want to set authentication to peer in pg_hba.conf or use a ~/.pgpass file (see Section 34.16).

8.
both servers: Make sure both database servers are stopped using, on Unix, e.g.:

```
pg_ctl -D /opt/PostgreSQL/9.6 stop
pg_ctl -D /opt/PostgreSQL/15 stop
```

or on Windows, using the proper service names:

```
NET STOP postgresql-9.6
NET STOP postgresql-15
```

Streaming replication and log-shipping standby servers must be running during this shutdown so they receive all changes.

9.
for standby server upgrades: If you are upgrading standby servers using methods outlined in section Step 11, verify that the old standby servers are caught up by running pg_controldata against the old primary and standby clusters. Verify that the "Latest checkpoint location" values match in all clusters. Also, make sure *wal_level* is not set to minimal in the postgresql.conf file on the new primary cluster.

10.
pg_upgrade: Always run the pg_upgrade binary of the new server, not the old one. pg_upgrade requires the specification of the old and new cluster's data and executable (bin) directories. You can also specify user and port values, and whether you want the data files linked or cloned instead of the default copy behavior.

If you use link mode, the upgrade will be much faster (no file copying) and use less disk space, but you will not be able to access your old cluster once you start the new cluster after the upgrade. Link mode also requires that the old and new cluster data directories be in the same file system. (Tablespaces and pg_wal can be on different file systems.) Clone mode provides the same speed and disk space advantages but does not cause the old cluster to be unusable once the new cluster is started. Clone mode also requires that the old and new data directories be in the same file system. This mode is only available on certain operating

systems and file systems.

The **--jobs** option allows multiple CPU cores to be used for copying/linking of files and to dump and restore database schemas in parallel; a good place to start is the maximum of the number of CPU cores and tablespaces. This option can dramatically reduce the time to upgrade a multi-database server running on a multiprocessor machine.

For Windows users, you must be logged into an administrative account, and then start a shell as the postgres user and set the proper path:

```
RUNAS /USER:postgres "CMD.EXE"
SET PATH=%PATH%;C:\Program Files\PostgreSQL\15\bin;
```

and then run pg_upgrade with quoted directories, e.g.:

```
pg_upgrade.exe
      --old-datadir "C:/Program Files/PostgreSQL/9.6/data"
      --new-datadir "C:/Program Files/PostgreSQL/15/data"
      --old-bindir "C:/Program Files/PostgreSQL/9.6/bin"
      --new-bindir "C:/Program Files/PostgreSQL/15/bin"
```

Once started, **pg_upgrade** will verify the two clusters are compatible and then do the upgrade. You can use **pg_upgrade --check** to perform only the checks, even if the old server is still running. **pg_upgrade --check** will also outline any manual adjustments you will need to make after the upgrade. If you are going to be using link or clone mode, you should use the option **--link** or **--clone** with **--check** to enable mode-specific checks. **pg_upgrade** requires write permission in the current directory.

Obviously, no one should be accessing the clusters during the upgrade. pg_upgrade defaults to running servers on port 50432 to avoid unintended client connections. You can use the same port number for both clusters when doing an upgrade because the old and new clusters will not be running at the same time. However, when checking an old running server, the old and new port numbers must be different.

If an error occurs while restoring the database schema, **pg_upgrade** will exit and you will have to revert to the old cluster as outlined in Step 17 below. To try **pg_upgrade** again, you will need to modify the old cluster so the pg_upgrade schema restore succeeds. If the problem is a contrib

module, you might need to uninstall the contrib module from the old cluster and install it in the new cluster after the upgrade, assuming the module is not being used to store user data.

11. streaming replication and log-shipping standby servers: If you used link mode and have Streaming Replication (see Section 27.2.5) or Log-Shipping (see Section 27.2) standby servers, you can follow these steps to quickly upgrade them. You will not be running pg_upgrade on the standby servers, but rather rsync on the primary. Do not start any servers yet.

If you did *not* use link mode, do not have or do not want to use rsync, or want an easier solution, skip the instructions in this section and simply recreate the standby servers once pg_upgrade completes and the new primary is running.

1. the new PostgreSQL binaries on standby servers: Make sure the new binaries and support files are installed on all standby servers.

2. sure the new standby data directories do *not* exist: Make sure the new standby data directories do *not* exist or are empty. If initdb was run, delete the standby servers' new data directories.

3. extension shared object files: Install the same extension shared object files on the new standbys that you installed in the new primary cluster.

4. standby servers: If the standby servers are still running, stop them now using the above instructions.

5. configuration files: Save any configuration files from the old standbys' configuration directories you need to keep, e.g., postgresql.conf (and any files included by it), postgresql.auto.conf, pg_hba.conf, because these will be overwritten or removed in the next step.

6. rsync: When using link mode, standby servers can be quickly upgraded using rsync. To accomplish this, from a directory on the primary server that is above the old and new database cluster directories, run this on the *primary* for each standby server:

```
rsync --archive --delete --hard-links --size-only --no-inc-recursive old_cluster new_cluster remote_dir
```

where **old_cluster** and **new_cluster** are relative to the current directory on the primary, and **remote_dir** is *above* the old and new cluster directories on the standby. The directory structure under the specified directories on the primary and standbys must match. Consult the rsync manual page for details on specifying the remote directory, e.g.,

```
rsync --archive --delete --hard-links --size-only --no-inc-recursive /opt/PostgreSQL/9.5 \
    /opt/PostgreSQL/9.6 standby.example.com:/opt/PostgreSQL
```

You can verify what the command will do using rsync's **--dry-run** option. While rsync must be run on the primary for at least one standby, it is possible to run rsync on an upgraded standby to upgrade other standbys, as long as the upgraded standby has not been started.

What this does is to record the links created by pg_upgrade's link mode that connect files in the old and new clusters on the primary server. It then finds matching files in the standby's old cluster and creates links for them in the standby's new cluster. Files that were not linked on the primary are copied from the primary to the standby. (They are usually small.) This provides rapid standby upgrades. Unfortunately, rsync needlessly copies files associated with temporary and unlogged tables because these files don't normally exist on standby servers.

If you have tablespaces, you will need to run a similar rsync command for each tablespace directory, e.g.:

```
rsync --archive --delete --hard-links --size-only --no-inc-recursive /vol1/pg_tblsp/PG_9.5_201510051 \
    /vol1/pg_tblsp/PG_9.6_201608131 standby.example.com:/vol1/pg_tblsp
```

If you have relocated pg_wal outside the data directories, rsync must be run on those directories too.

7.

streaming replication and log-shipping standby servers: Configure the servers for log shipping. (You do not need to run **pg_backup_start()** and **pg_backup_stop()** or take a file system backup as the standbys are still synchronized with the primary.) Replication slots are not copied and must be recreated.

12.

pg_hba.conf: If you modified pg_hba.conf, restore its original settings. It might also be necessary to adjust other configuration files in the new cluster to match the old cluster, e.g., postgresql.conf (and any files included by it), postgresql.auto.conf.

13.
the new server: The new server can now be safely started, and then any rsync'ed standby servers.

14.
processing: If any post-upgrade processing is required, pg_upgrade will issue warnings as it completes. It will also generate script files that must be run by the administrator. The script files will connect to each database that needs post-upgrade processing. Each script should be run using:

```
psql --username=postgres --file=script.sql postgres
```

The scripts can be run in any order and can be deleted once they have been run.

Caution
In general it is unsafe to access tables referenced in rebuild scripts until the rebuild scripts have run to completion; doing so could yield incorrect results or poor performance. Tables not referenced in rebuild scripts can be accessed immediately.

15.
Because optimizer statistics are not transferred by **pg_upgrade**, you will be instructed to run a command to regenerate that information at the end of the upgrade. You might need to set connection parameters to match your new cluster.

16.
old cluster: Once you are satisfied with the upgrade, you can delete the old cluster's data directories by running the script mentioned when **pg_upgrade** completes. (Automatic deletion is not possible if you have user-defined tablespaces inside the old data directory.) You can also delete the old installation directories (e.g., bin, share).

17.
to old cluster: If, after running **pg_upgrade**, you wish to revert to the old cluster, there are several options:

- the **--check** option was used, the old cluster was unmodified; it can be restarted.

- the **--link** option was *not* used, the old cluster was unmodified; it can be restarted.

- the **--link** option was used, the data files might be shared between the old and new cluster:
  - **pg_upgrade** aborted before linking started, the old cluster was unmodified; it can be restarted.
  - you did *not* start the new cluster, the old cluster was unmodified except that, when linking started, a .old suffix was appended to $PGDATA/global/pg_control. To reuse the old cluster, remove the .old suffix from $PGDATA/global/pg_control; you can then restart the old cluster.
  - you did start the new cluster, it has written to shared files and it is unsafe to use the old cluster. The old cluster will need to be restored from backup in this case.

NOTES

pg_upgrade creates various working files, such as schema dumps, stored within pg_upgrade_output.d in the directory of the new cluster. Each run creates a new subdirectory named with a timestamp formatted as per ISO 8601 (%Y%m%dT%H%M%S), where all its generated files are stored. pg_upgrade_output.d and its contained files will be removed automatically if pg_upgrade completes successfully; but in the event of trouble, the files there may provide useful debugging information.

pg_upgrade launches short-lived postmasters in the old and new data directories. Temporary Unix socket files for communication with these postmasters are, by default, made in the current working directory. In some situations the path name for the current directory might be too long to be a valid socket name. In that case you can use the **-s** option to put the socket files in some directory with a shorter path name. For security, be sure that that directory is not readable or writable by any other users. (This is not supported on Windows.)

All failure, rebuild, and reindex cases will be reported by pg_upgrade if they affect your installation; post-upgrade scripts to rebuild tables and indexes will be generated automatically. If you are trying to automate the upgrade of many clusters, you should find that clusters with identical database schemas require the same post-upgrade steps for all cluster upgrades; this is because the post-upgrade steps are based on the database schemas, and not user data.

For deployment testing, create a schema-only copy of the old cluster, insert dummy data, and upgrade that.

pg_upgrade does not support upgrading of databases containing table columns using these reg* OID-referencing system data types:

regcollation
regconfig
regdictionary
regnamespace
regoper
regoperator
regproc
regprocedure

(regclass, regrole, and regtype can be upgraded.)

If you want to use link mode and you do not want your old cluster to be modified when the new cluster is started, consider using the clone mode. If that is not available, make a copy of the old cluster and upgrade that in link mode. To make a valid copy of the old cluster, use **rsync** to create a dirty copy of the old cluster while the server is running, then shut down the old server and run **rsync --checksum** again to update the copy with any changes to make it consistent. (**--checksum** is necessary because **rsync** only has file modification-time granularity of one second.) You might want to exclude some files, e.g., postmaster.pid, as documented in Section 26.3.3. If your file system supports file system snapshots or copy-on-write file copies, you can use that to make a backup of the old cluster and tablespaces, though the snapshot and copies must be created simultaneously or while the database server is down.

SEE ALSO

initdb(1), **pg_ctl**(1), **pg_dump**(1), **postgres**(1)