

*****Backup and Restore*****

`pg_dump` and `pg_dumpall` are two utilities provided by PostgreSQL for creating logical backups of databases.

1. Logical Backup:-

- `pg_dump`, `pg_dumpall` and `pg_restore` utilities are used for logical backups.
- `pg_dump` will help in taking backups for Database level, Schema level and Table Level.
- `Pg_dumpall` used for cluster level dump.

A. `pg_dump`:-

- `pg_dump` will help in taking backups for Database level, Schema level and Table Level.
- `pg_dump` is used to back up a single PostgreSQL database into a script or archive file. The backup can be restored later using `psql` or `pg_restore`.

Syntax:-

`Pg_dump -U user_name -d database_name > location(/home/gaurav)`

Step01:- Make directory and owner of this directory is postgres

```
[root@standby /]# mkdir backup
[root@standby /]# chown 'postgres:postgres' backup
```

Step 02:-

```
[root@standby /]# su - postgres
```

Step 03:-

```
bash-4.2$ pg_dump -U postgres -d gaurav > /backup/pgbackup
bash-4.2$ logout
```

```
$ pg_dump -U username -Fp dbname > filename.txt
$ pg_dump -U username dbname -f filename.txt
$ pg_dump -Fp -U username dbname -f filename.txt
```

-U =user_name

-d=databasename

-F = format

1. Fp =plain

2. Fc=custom

3. Ft = tar

1. Plain SQL Script:

```
pg_dump -U username -W -F p -f output_file.sql database_name
```

- `-U username``: Specifies the database user.
- `-W``: Prompts for the password.
- `-F p``: Specifies the format as plain SQL.
- `-f output_file.sql``: Output file.
- `database_name``: Name of the database to back up.

2. Custom Archive Format:

```
pg_dump -U username -W -F c -f output_file.dump database_name
```

- `-F c``: Specifies the format as a custom archive.
- Custom format allows for more flexible restoration options using `pg_restore``.

3. Directory Format:

```
pg_dump -U username -W -F d -f output_directory database_name
```

- `-F d``: Specifies the format as a directory.
- The output is a directory with one file per table, allowing for parallel restoration.

4. Compressing Output:

```
pg_dump -U username -W -Fc -Z 9 -f output_file.dump database_name
```

- `-Z 9``: Compresses the output with maximum compression.

5. Backing Up Specific Tables:

```
pg_dump -U username -W -Fc -t table_name -d database_name -f output_file.dump
```

- `-t table_name``: Specifies a table to back up.

6. Exclude ownership and Privilage

```
pg_dump -U smile_mod -h 10.189.8.160 -p 54032 -d polc_claims_db --schema  
claims -v -x -O -f /backup_25/calmis.sql > backup.log 2>&1
```

- `-v` :- Verbose mode — shows detailed progress output.
- `-x` :- Exclude privileges (GRANT/REVOKE) from the dump.

- O :- Exclude ownership information (i.e., no OWNER TO ... lines).
- Fc :- Dump in custom format, useful for selective restoration via pg_restore.

7. Parallel cpu:-

```
pg_dump -U postgres -h localhost -d mydatabase -Fd -j 4 -Z 9 -f  
mydatabase_backup_dir
```

- -j 4 Use 4 parallel jobs to speed up the dump (only valid with -Fd).
- -Z 9 Use maximum compression (level 9).

8. Exclude tables backup: -

Option:

-T: Do not dump any tables matching pattern.

- **Exclude single Table :-**

```
pg_dump -U user_name -d db_name -T table_name -p 5432 -f  
/backup_path/exclude_tablename_bkp.sql
```

- **Exclude multiple tables:-**

```
pg_dump -U user_name -d db_name -T table_name1 -T table_name2 -p 5432 -f  
/backup_path/exclude_multi_tables_bkp.sql
```

- **Exclude tables named with matching pattern backup :-**

```
pg_dump -U user_name -d db_name -T table_name* bkp -p 5432 -f  
/backup_path/exclude_matching_tables_bkp.sql
```

9. Schema only backup: -

Option:

-s: Dump only the object definitions (schema), not data.

```
pg_dump -U user_name -d db_name -s -p 5432 -f /backup_path/Schema_bkp.sql
```

10. Data only backup:-

Option:

-a:

- Dump only the data, not the schema (data definitions). Table data, large objects, and sequence values are dumped.
- Only the table data (rows) will be exported.

- No table structures, sequences, indexes, or constraints will be included.
- Essentially, it will create INSERT statements for every row in all tables of db_name.
- Exclude all schema definitions (like CREATE TABLE, CREATE INDEX, ALTER TABLE, etc.)
- Include only INSERT or COPY commands that load data into existing tables.

`pg_dump -U user_name -d db_name -a -p 5432 -f /backup_path/Data_bkp.sql`

11. Compress backup file

Syntax:-

`Pg_dump -U postgres -d gaurav | gzip > /backup/pgback`

=====

B. pg_dumpall:-

- It is used to take backup of entire cluster
- `pg_dumpall` is used to back up an entire PostgreSQL database cluster, including all databases, roles, and tablespaces.

Step 01:-

```
[root@standby /]# mkdir backup
[root@standby /]# chown 'postgres:postgres' backup
```

Step 02:-

```
[root@standby /]# su - postgres
```

Step 03:-

```
-bash-4.2$ pg_dumpall -U postgres > /backup/all
```

1. Plain SQL Script:

`pg_dumpall -U username -W -f output_file.sql`

— Backs up all databases, roles, and tablespaces into a single plain SQL script file.

— This file can be restored using `psql`.

2. Backing Up Globals Only:

Option:

-g: Dump only global objects (roles and tablespaces), no databases.

```
pg_dumpall -U user_name -g -p 5432 -f /backup_path/global_objects_bkp.sql
```

3. Dump only tablespaces :-

Option:

-t: Dump only tablespaces, no databases, or roles.

```
pg_dumpall -U user_name -t -p 5432 -f /backup_path/tablespaces_bkp.sql
```

❖ Handling Large Databases: :-

Some operating systems have maximum file size limits that cause problems when creating large pg_dump output files. Fortunately, pg_dump can write to the standard output, so you can use standard Unix tools to work around this potential problem.

#Use compressed dump

```
pg_dump -U user_name -d db_name -p 5432 | gzip -c > /backup_path/backup_filename.sql.gz
```

#Restore with

```
zcat /backup_path/backup_filename.sql.gz | psql -U user_name -d db_name -p 5432
```

Differences Between pg_dump ,pg_dumpall and pg_basebackup

1. Scope:

- `pg_dump`: Backs up a single database.
- `pg_dumpall`: Backs up all databases in a cluster, including roles and tablespaces.

2. Output Formats:

- `pg_dump`: Supports multiple output formats (plain, custom, directory, tar).
- `pg_dumpall`: Only outputs plain SQL scripts.

3. Restoration:

- `pg_dump`: The backup can be restored using `psql` (for plain SQL format) or `pg_restore` (for other formats).
- `pg_dumpall`: The backup is restored using `psql`.

4. Use Cases:

- `pg_dump`: Used for backing up individual databases, especially when specific customization or flexibility in restoration is required.
- `pg_dumpall`: Used for comprehensive backups of the entire PostgreSQL instance, including all databases, roles, and tablespaces.

=====

********Restore (pg_restore or psql)*******

- If your dumps(backup) are Custom or Tar Format you need to use only **pg_restore** utility
- If your dumps(backup) are Plain SQL format you need to use **psql** utility

Syntax:-

1. psql:-

Psql -U postgres -d jk < /backup/pgbackup

```
CREATE DATABASE
postgres=# \q
-bash-4.2$ psql -U postgres -d jk < /backup/pgbackup
psql
```

It is used when backup file in plain format

2. pg_restore:-

```
all bk.tar pgbackup tar
-bash-4.2$ pg_restore -U postgres -d kh /backup/bk.tar
```

It is used when backup in tar format

=====

*******pg_basebackup*******

- Pg_basebackup is used to take the base backup of running PostgreSQL database cluster.
- This backup can be used for PITR or replication.
- Bckups are always taken of the entire cluster and cannot be used for single database or objects

Syntax:-

Pg_basebackup -h <ip address> -p <port number> -U <user_name> -D <directory location> -Ft -Xs -R -P

-h =hostname

-p = port number

-U =username

-D = directory location

-F = format

-Xs = wal method - stream

-R = write configuration parameter for replication

-P = postures information

```
-bash-4.2$ pg_basebackup -U postgres -D /backup/pgbase -Fp -P -Xs -R
53283/53283 kB (100%), 1/1 tablespace
-bash-4.2$ cd /
```

3. Backup command :-

```
/usr/pgsql-15/bin/pg_basebackup -U postgres -p 5432 -D
$backup_dir/PostgreSQL_Base_Backup_$(date +"%d-%m-%y-%H%M%S") -l "`date`" -P -Ft -R -z
```

-U postgres → Use PostgreSQL user postgres.

-h /tmp → Connect to PostgreSQL using the Unix socket in /tmp.

-p 4702 → Port number = 4702.

-D \$backup_dir/... → Destination directory for the backup.

Here it's timestamped → PostgreSQL_Base_Backup_DD-MM-YY-HHMMSS.

-l "\$(date)" → Backup label, will be current date/time.

-P → Show progress during backup.

-Ft → Write output as a tar archive.

-z → Compress the backup with gzip.

-R → Write a standby.signal file and a correct postgresql.auto.conf with primary_conninfo.

This makes the backup ready to start as a replica (standby server).

4. controll bandwidth:-

```
pg_basebackup -h <server_ip> -U <replication_user> -D /backup_dir --progress --checkpoint-  
timeout=300
```

5. Replication:-

With checkpoint option

```
pg_basebackup -D /data/pgdata --checkpoint=fast -h 100.127.129.45 -p 5432 -Xs -R -P > backup.log  
2>&1
```

Without option

```
pg_basebackup -D /data/pgdata -h 100.127.129.45 -p 5432 -Xs -R -P > backup.log 2>&1
```

-D /var/lib/pgsql/15/data → Target data directory for restore.

-h 192.168.29.193 → Source PostgreSQL server IP.

-p 5432 → Port number.

-Xs → Include WAL files in tar or plain format.

-R → Automatically write standby.signal and primary_conninfo (for streaming replication).

-P → Show progress.

> backup.log 2>&1 → Logs all output (stdout + stderr) to backup.log.

```
*****role*****
```

The following command will export the roles only:-

```
Pg_dumpall --roles-only -U postgres > all_db_roles.sql
```

The following command will export schemas only:-

```
Pg_dumpall --schema-only -U postgres > all_db_schema.sql
```

The following command will export only the tablespace definitions:-

```
Pg_dumpall --tablespaces-only -U postgres > all_db_tablespaces.sql
```

******* Backup and Restortion******

1. First we take full backup on server:- 20.243.32.40 With the help pg_dumpall command.
Command:- pg_dumpall -U postgres > full.bkp.sql
2. Than we transfer this file on destination server 40.81.213.8.
3. We restore this file with help psql command.

Command: - psql -U postgres < full_bkp.sql

Incremental backup:-

Step 1. Enable WAL Summary

summarize_wal = 'on' in postgresql.conf.

Step 2. Take a full backup

```
/usr/lib/pgsql-17.0/bin/pg_basebackup -D /backup/fullbackup/ -c fast -p 5432 -v
```

Step 3. Take a 1st incremental backup with full backup manifest

```
/usr/lib/pgsql-17.0/bin/pg_basebackup -D /backup/employees_db_incremental_1stbackup -i  
/backup/backup_manifest -c fast -p 5432 -v
```

Step 4. Take a 2nd incremental backup with full backup manifest

```
/usr/lib/pgsql-17.0/bin/pg_basebackup -D /backup/employees_db_incremental_2ndbackup -i  
/backup/employees_db_incremental_1stbackup/backup_manifest -c fast -p 5432 -v
```

Step 5. Combine All Backups in one specific directory

```
/usr/lib/pgsql-17.0/bin/pg_combinebackup -o /abc /backup  
/backup/employees_db_incremental_1stbackup  
/backup/employees_db_incremental_2ndbackup
```

Step 6. Edit combine backup file Config and change port then Start Server

```
/usr/lib/pgsql-17.0/bin/pg_ctl -D /abc start
```

Step 7. Validate Recovery

```
psql -U postgres incr_backup -p 5433
```

```
SELECT * FROM employees;
```

Prodction database sceniores:-

1. Backup (pg_dump):-

```
pg_dump --file "/home/pgsql/backup/$backupFileName" \  
--host $SOURCE_PGHOST \  
--port $SOURCE_PGPORT \  
--username $SOURCE_PGUSER \  
--verbose \  
--format=c \  
--blobs \  
--data-only \  
--no-owner \  
--no-privileges \  
--no-tablespaces \  
-d ${db} \  
> "/home/pgsql/backup/log/$backup_log" 2>&1
```

✔Explanation:

- --file → where to save the dump.
- --host, --port, --username → connection info.
- --verbose → prints details of progress.
- --format=c → custom format (recommended, can be used with pg_restore).

- --blobs → includes large objects (BLOBs).
- --data-only → only data, no schema.
- --no-owner, --no-privileges, --no-tablespaces → ignore ownership, GRANTs, tablespaces.
- -d \${db} → the source database.
- > ...log 2>&1 → write output and errors to a log file.

☐ So this step takes a data-only backup of the source DB into a custom-format dump file.

☐ 2. Truncate data in target database:-

```
psql -h 172.16.34.25 -p 5432 -U sa_ecgc -d poluw_shd_rod_lossmin_db \
-c "DO \$\$
DECLARE row record;
BEGIN
FOR row IN
SELECT table_schema, table_name
FROM information_schema.tables
WHERE table_schema NOT IN ('information_schema', 'pg_catalog')
AND table_schema NOT LIKE 'pg_toast%'
AND table_schema NOT LIKE 'pg_temp%'
AND table_type = 'BASE TABLE'
LOOP
EXECUTE 'TRUNCATE TABLE ' || quote_ident(row.table_schema) || '.' ||
quote_ident(row.table_name) || ' CASCADE;';
END LOOP;
END
\$\$;"
```

✔Explanation:

- Connects to target database (poluw_shd_rod_lossmin_db).
- Runs a PL/pgSQL DO block that:
 1. Finds all user tables (ignores system schemas like pg_catalog, information_schema, toast, temp).
 2. Iterates through them.
 3. Runs TRUNCATE TABLE ... CASCADE on each → removes all rows from all tables.
- Effect: the target DB is emptied but schema (tables, constraints, indexes, functions) remain intact.

☐ 3. Restore (pg_restore):-

```
pg_restore \
--host $TARGET_PGHOST \
--port $TARGET_PGPORT \
```

```
--username $TARGET_PGUSER \
--dbname ${db} \
--verbose \
--no-owner \
--no-privileges \
--no-tablespaces \
--disable-triggers \
--data-only\
    /home/pgsql/backup/ecib_uw_db_29_09_2025_00_15_01_only_data.sql \
> /home/pgsql/restore.log 2>&1
```

✓ Explanation:

Connects to target DB: ecib_uw_db.

- --clean → drops database objects before recreating them (in this case it will run TRUNCATE or DELETE on tables before loading data, since it's a data-only dump).

● 📌 Normally (schema + data dump)

1. If you run pg_restore --clean on a full dump (schema + data):
2. --clean tells pg_restore to drop existing objects first (DROP TABLE, DROP VIEW, DROP FUNCTION, etc.)
3. Then it recreates the schema from the dump, and finally loads the data.

📌 So the target DB is made to look exactly like the backup.

● 📌 In your case (data-only dump)

1. Since your dump is created with --data-only, the file does not contain CREATE TABLE / CREATE VIEW / CREATE FUNCTION statements.
2. It only has the data (INSERT statements or COPY commands).

So, when you run:

pg_restore --clean --data-only ...

- pg_restore will still try to clean before restore.
- But since there are no schema objects in the dump, there are no DROP TABLE statements in it.
- Instead, the "cleaning" step applies to the tables you're loading data into.

It means pg_restore will issue a TRUNCATE or DELETE on the table before inserting rows, to ensure you don't get duplicates.

- --verbose → detailed output.

- `--no-tablespaces` → ignores tablespace info.
- `--disable-triggers` → disables foreign key triggers during restore (avoids FK violations while loading). They are re-enabled after restore.

`/home/pgsql/backup/...sql` → the dump file you created.

- `> restore.log 2>&1` → logs everything.
- `--no-privileges` → don't restore GRANT/REVOKE statements.
- `--no-owner` → don't restore object ownership (keeps target DB's ownerships as is).

🔗 This step restores the dumped data into the target database. Since schema already exists, only data is inserted.