

Postgres Database

- [Postgres Database Manager](#)
- [Install ppostgress in Debian and Ubuntu Linux](#)
- [How to change the password of the user postgres](#)

Postgres Database Manager

This is a script to manage the postgres database.

pgmanager.py

```
#!/usr/bin/env python3
"""
PostgreSQL Interactive CLI Manager
-----
A menu-driven tool to manage PostgreSQL databases, users, and transactions.

Requirements:
    pip install psycopg2-binary

Usage:
    python pg_manager.py
"""

import sys
import getpass
from typing import Optional

try:
    import psycopg2
    from psycopg2 import sql
    from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
except ImportError:
    print("Missing dependency. Install it with: pip install psycopg2-binary")
    sys.exit(1)

# ----- ANSI colors for nicer output -----
class C:
    RESET = "\033[0m"
    BOLD = "\033[1m"
    DIM = "\033[2m"
    RED = "\033[91m"
```

```
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
MAGENTA = "\033[95m"
CYAN = "\033[96m"
```

```
def banner(text: str) -> None:
    line = "=" * (len(text) + 4)
    print(f"\n{C.CYAN}{C.BOLD}{line}\n {text}\n{line}{C.RESET}")
```

```
def info(msg: str) -> None:
    print(f"{C.BLUE}[i]{C.RESET} {msg}")
```

```
def ok(msg: str) -> None:
    print(f"{C.GREEN}[✓]{C.RESET} {msg}")
```

```
def warn(msg: str) -> None:
    print(f"{C.YELLOW}[!]{C.RESET} {msg}")
```

```
def err(msg: str) -> None:
    print(f"{C.RED}[x]{C.RESET} {msg}")
```

```
def prompt(msg: str, default: Optional[str] = None) -> str:
    suffix = f" [{default}]" if default else ""
    val = input(f"{C.BOLD}{msg}{suffix}: {C.RESET}").strip()
    return val or (default or "")
```

```
def confirm(msg: str) -> bool:
    return prompt(f"{msg} (y/N)").lower() in ("y", "yes")
```

```
def print_table(headers, rows) -> None:
    """Render a simple aligned table."""
```

```

if not rows:
    info("(no rows)")
    return
widths = [len(str(h)) for h in headers]
for row in rows:
    for i, cell in enumerate(row):
        widths[i] = max(widths[i], len(str(cell)) if cell is not None else 4)

def fmt_row(cells):
    return " | ".join(
        str(" " if c is None else c).ljust(widths[i]) for i, c in enumerate(cells)
    )

sep = "--".join("-" * w for w in widths)
print(f"{C.BOLD}{fmt_row(headers)}{C.RESET}")
print(sep)
for row in rows:
    print(fmt_row(row))
print(f"{C.DIM}({len(rows)} row{'s' if len(rows) != 1 else ''}){C.RESET}")

```

----- Connection management -----

```

class PGManager:
    def __init__(self):
        self.host = "localhost"
        self.port = 5432
        self.user = "postgres"
        self.password = ""
        self.current_db = "postgres"

    def connect(self, dbname: Optional[str] = None, autocommit: bool = False):
        """Open a new connection to the server / a specific database."""
        conn = psycopg2.connect(
            host=self.host,
            port=self.port,
            user=self.user,
            password=self.password,
            dbname=dbname or self.current_db,
        )
        if autocommit:

```

```

        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
    return conn

def setup_connection(self):
    """Collect connection details from the user."""
    banner("PostgreSQL Connection Setup")
    self.host = prompt("Host", "localhost")
    port_str = prompt("Port", "5432")
    try:
        self.port = int(port_str)
    except ValueError:
        warn("Invalid port, falling back to 5432")
        self.port = 5432
    self.user = prompt("User", "postgres")
    self.password = getpass.getpass(f"{C.BOLD}Password (hidden): {C.RESET}")
    self.current_db = prompt("Initial database", "postgres")

# Test the connection
try:
    conn = self.connect()
    with conn.cursor() as cur:
        cur.execute("SELECT version();")
        version = cur.fetchone()[0]
    conn.close()
    ok("Connected successfully.")
    info(version)
except psycopg2.Error as e:
    err(f"Connection failed: {e}")
    sys.exit(1)

# ----- Operations -----
def list_databases(self):
    banner("Databases")
    conn = self.connect("postgres")
    try:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT d.datname AS name,
                       pg_catalog.pg_get_userbyid(d.datdba) AS owner,
                       pg_catalog.pg_encoding_to_char(d.encoding) AS encoding,

```

```

        pg_size_pretty(pg_database_size(d.datname)) AS size
    FROM pg_catalog.pg_database d
    WHERE NOT d.datistemplate
    ORDER BY d.datname;
    """
    rows = cur.fetchall()
    print_table(["Database", "Owner", "Encoding", "Size"], rows)
    return [r[0] for r in rows]
finally:
    conn.close()

def select_database(self):
    banner("Select Database")
    dbs = self.list_databases()
    if not dbs:
        warn("No databases available.")
        return
    print()
    for i, db in enumerate(dbs, 1):
        marker = f" {C.GREEN}(current){C.RESET}" if db == self.current_db else ""
        print(f"  {C.BOLD}{i}{C.RESET} {db}{marker}")
    choice = prompt("\nPick a database number (or press Enter to cancel)")
    if not choice:
        return
    try:
        idx = int(choice) - 1
        if 0 <= idx < len(dbs):
            self.current_db = dbs[idx]
            ok(f"Now working with: {self.current_db}")
        else:
            err("Out of range.")
    except ValueError:
        err("Not a valid number.")

def list_tables(self):
    banner(f"Tables in '{self.current_db}'")
    try:
        conn = self.connect()
    except psycopg2.Error as e:
        err(f"Cannot connect to {self.current_db}: {e}")

```

```

        return
    try:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT schemaname, tablename, tableowner,
                       pg_size_pretty(
                           pg_total_relation_size(
                               quote_ident(schemaname) || '.' || quote_ident(tablename)
                           )
                       ) AS size
                FROM pg_catalog.pg_tables
                WHERE schemaname NOT IN ('pg_catalog', 'information_schema')
                ORDER BY schemaname, tablename;
            """)
            rows = cur.fetchall()
            print_table(["Schema", "Table", "Owner", "Size"], rows)
    finally:
        conn.close()

```

```

def view_transactions(self):
    banner("Active Sessions & Transactions")
    info("Showing pg_stat_activity across all databases\n")
    conn = self.connect("postgres")
    try:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT pid,
                       datname,
                       username,
                       state,
                       COALESCE(EXTRACT(EPOCH FROM (now() - xact_start))::int, 0)
                          AS xact_age_s,
                       COALESCE(LEFT(query, 60), '') AS query
                FROM pg_stat_activity
                WHERE backend_type = 'client backend'
                ORDER BY datname, state, pid;
            """)
            rows = cur.fetchall()
            print_table(
                ["PID", "Database", "User", "State", "Txn Age (s)", "Query (truncated)"],

```

```

        rows,
    )
finally:
    conn.close()

def create_database(self):
    banner("Create Database")
    name = prompt("New database name")
    if not name:
        warn("Cancelled.")
        return
    owner = prompt("Owner (leave empty for current user)", "")
    conn = self.connect("postgres", autocommit=True) # CREATE DATABASE needs autocommit
    try:
        with conn.cursor() as cur:
            if owner:
                cur.execute(
                    sql.SQL("CREATE DATABASE {} OWNER {}".format(
                        sql.Identifier(name), sql.Identifier(owner)
                    )
                )
            else:
                cur.execute(
                    sql.SQL("CREATE DATABASE {}".format(sql.Identifier(name)))
                )
            ok(f"Database '{name}' created.")
    except psycopg2.Error as e:
        err(f"Failed: {e}")
    finally:
        conn.close()

def create_user(self):
    banner("Create User / Role")
    name = prompt("New username")
    if not name:
        warn("Cancelled.")
        return
    pwd = getpass.getpass(f"{C.BOLD}Password for {name} (hidden): {C.RESET}")
    if not pwd:
        warn("Password cannot be empty.")

```

```

        return
    is_super = confirm("Should this user be a SUPERUSER?")
    can_create_db = confirm("Allow CREATEDB?")

    conn = self.connect("postgres", autocommit=True)
    try:
        opts = []
        if is_super:
            opts.append(sql.SQL("SUPERUSER"))
        if can_create_db:
            opts.append(sql.SQL("CREATEDB"))
        opts_sql = sql.SQL(" ").join(opts) if opts else sql.SQL("")

        with conn.cursor() as cur:
            cur.execute(
                sql.SQL("CREATE USER {} WITH PASSWORD %s {}".format(
                    sql.Identifier(name), opts_sql
                )),
                [pwd],
            )
            ok(f"User '{name}' created.")
    except psycopg2.Error as e:
        err(f"Failed: {e}")
    finally:
        conn.close()

def assign_user_to_database(self):
    banner("Grant User Access to a Database")
    dbs = self.list_databases()
    if not dbs:
        return
    print()
    for i, db in enumerate(dbs, 1):
        print(f"  {C.BOLD}{i}{C.RESET} {db}")
    db_choice = prompt("\nPick a database number")
    try:
        db_name = dbs[int(db_choice) - 1]
    except (ValueError, IndexError):
        err("Invalid selection.")
    return

```

```

username = prompt("Username to grant access")
if not username:
    warn("Cancelled.")
    return

print(f"""
{C.BOLD}Privilege level:{C.RESET}
1) CONNECT only (can connect, no data access)
2) Read-only on existing tables in public schema
3) Read/Write on existing tables in public schema
4) ALL PRIVILEGES on the database (full admin on this DB)
5) Make user the OWNER of the database
""")

level = prompt("Choose 1-5")

try:
    if level == "5":
        # Change owner requires a connection to any DB with sufficient privs
        conn = self.connect("postgres", autocommit=True)
        with conn.cursor() as cur:
            cur.execute(
                sql.SQL("ALTER DATABASE {} OWNER TO {}".format(
                    sql.Identifier(db_name), sql.Identifier(username)
                )
            )
            conn.close()
            ok(f"'{username}' is now OWNER of '{db_name}'")
            return

        # First grant CONNECT (and possibly DB-level privileges)
        conn = self.connect("postgres", autocommit=True)
        with conn.cursor() as cur:
            if level == "4":
                cur.execute(
                    sql.SQL("GRANT ALL PRIVILEGES ON DATABASE {} TO {}".format(
                        sql.Identifier(db_name), sql.Identifier(username)
                    )
                )
            )
        else:

```

```

        cur.execute(
            sql.SQL("GRANT CONNECT ON DATABASE {} TO {}").format(
                sql.Identifier(db_name), sql.Identifier(username)
            )
        )
    conn.close()

# For table-level grants, connect to the target DB
if level in ("2", "3"):
    conn = self.connect(db_name, autocommit=True)
    with conn.cursor() as cur:
        cur.execute(
            sql.SQL("GRANT USAGE ON SCHEMA public TO {}").format(
                sql.Identifier(username)
            )
        )
    if level == "2":
        cur.execute(
            sql.SQL(
                "GRANT SELECT ON ALL TABLES IN SCHEMA public TO {}"
            ).format(sql.Identifier(username))
        )
        cur.execute(
            sql.SQL(
                "ALTER DEFAULT PRIVILEGES IN SCHEMA public "
                "GRANT SELECT ON TABLES TO {}"
            ).format(sql.Identifier(username))
        )
    else:
        cur.execute(
            sql.SQL(
                "GRANT SELECT, INSERT, UPDATE, DELETE "
                "ON ALL TABLES IN SCHEMA public TO {}"
            ).format(sql.Identifier(username))
        )
        cur.execute(
            sql.SQL(
                "ALTER DEFAULT PRIVILEGES IN SCHEMA public "
                "GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO {}"
            ).format(sql.Identifier(username))
        )

```

```

        )
        conn.close()

        ok(f"Privileges granted to '{username}' on '{db_name}'.")
    except psycopg2.Error as e:
        err(f"Failed: {e}")

def list_users(self):
    banner("Users / Roles")
    conn = self.connect("postgres")
    try:
        with conn.cursor() as cur:
            cur.execute("""
                SELECT rolname,
                       rolsuper,
                       rolcreatedb,
                       rolcreaterole,
                       rolcanlogin
                FROM pg_roles
                WHERE rolname NOT LIKE 'pg\\_%' ESCAPE '\\'
                ORDER BY rolname;
            """)
            rows = cur.fetchall()
            print_table(
                ["Role", "Superuser", "CreateDB", "CreateRole", "Can Login"], rows
            )
    finally:
        conn.close()

```

```
# ----- Menu -----
```

```
MENU = f"""
```

```

{C.BOLD}{C.MAGENTA}=== PostgreSQL Manager ==={C.RESET}
{C.BOLD}1){C.RESET} List databases
{C.BOLD}2){C.RESET} Select current database
{C.BOLD}3){C.RESET} List tables in current database
{C.BOLD}4){C.RESET} View ongoing transactions / sessions
{C.BOLD}5){C.RESET} Create new database
{C.BOLD}6){C.RESET} Create new user
{C.BOLD}7){C.RESET} Assign user to a database (grant privileges)

```

```
{C.BOLD}8){C.RESET} List users / roles
{C.BOLD}9){C.RESET} Reconnect (change credentials)
{C.BOLD}0){C.RESET} Exit
```

```
"""
```

```
def main():
    mgr = PGManager()
    mgr.setup_connection()

    actions = {
        "1": mgr.list_databases,
        "2": mgr.select_database,
        "3": mgr.list_tables,
        "4": mgr.view_transactions,
        "5": mgr.create_database,
        "6": mgr.create_user,
        "7": mgr.assign_user_to_database,
        "8": mgr.list_users,
        "9": mgr.setup_connection,
    }

    while True:
        print(MENU)
        print(
            f"{C.DIM}Current connection: "
            f"{mgr.user}@{mgr.host}:{mgr.port}/{mgr.current_db}{C.RESET}"
        )
        choice = prompt("Choose an option")
        if choice == "0":
            info("Goodbye.")
            break
        action = actions.get(choice)
        if action:
            try:
                action()
            except KeyboardInterrupt:
                warn("\nInterrupted.")
            except Exception as e:
                err(f"Unexpected error: {e}")
```

```
        else:
            warn("Unknown option.")

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print()
        info("Interrupted. Bye.")
```

Here's how to use it:

bash

```
pip install psycopg2-binary
python pg_manager.py
```

On launch it asks for host/port/user/password and tests the connection. Then you get a menu:

1. **List databases** — name, owner, encoding, and size
2. **Select current database** — pick from a numbered list; everything table-related uses this
3. **List tables** — schemas, tables, owners, sizes (skips system schemas)
4. **View ongoing transactions** — queries `pg_stat_activity` showing PID, db, user, state, transaction age, and the running query
5. **Create database** — optionally assign an owner
6. **Create user** — password prompt is hidden; can flag SUPERUSER / CREATEDB
7. **Assign user to a database** — five privilege levels: CONNECT only, read-only, read/write, ALL PRIVILEGES, or make them the owner. For read/write levels it also sets default privileges so future tables are covered.
8. **List users/roles** — with their attribute flags
9. **Reconnect** — switch to a different server or user without restarting

A few implementation notes worth knowing:

- All identifiers go through `psycopg2.sql.Identifier`, so database/user names are properly quoted and SQL-injection safe.
- `CREATE DATABASE` and `CREATE USER` need autocommit mode — they can't run inside a transaction block — so the script opens those connections with `ISOLATION_LEVEL_AUTOCOMMIT`.
- For table-level grants, it connects to the target database itself (you can't grant on `public` in db X while connected to db Y).
- The transactions view filters to `client backend` so you don't see autovacuum/walwriter noise.

If you want me to add anything — killing a stuck PID (`pg_terminate_backend`), revoking privileges, dumping schemas, or letting you run arbitrary SQL from the menu — say the word.

Install ppostgres in Debian and Ubuntu Linux

On Linux, after a fresh install (`apt install postgresql`, `dnf install postgresql-server`, etc.):

- **Default user:** `postgres` — both as a Linux system user and as the PostgreSQL superuser role.
- **Default password:** none. There is no preset password.

By default the `postgres` role uses **peer authentication** for local connections, which means PostgreSQL trusts the OS user. So you log in like this:

bash

```
sudo -u postgres psql
```

No password prompt — it works because you're acting as the Linux `postgres` user, and PostgreSQL matches that to the `postgres` role.

If you want to use the script I just made (which connects over TCP with a password), you need to **set a password** first:

bash

```
sudo -u postgres psql
```

Then inside `psql`:

sql

```
ALTER USER postgres WITH PASSWORD 'your_password_here';  
\q
```

You may also need to allow password logins over TCP. Edit `pg_hba.conf` (usually at `/etc/postgresql/<version>/main/pg_hba.conf` on Debian/Ubuntu, or `/var/lib/pgsql/data/pg_hba.conf` on RHEL/Fedora) and find lines like:

```
local  all  postgres  peer  
host   all  all       127.0.0.1/32  ident
```

Change the method to `scram-sha-256` (modern) or `md5` (older):

```
local  all  postgres  scram-sha-256
host   all  all         127.0.0.1/32  scram-sha-256
```

Then reload:

bash

```
sudo systemctl reload postgresql
```

After that, `psql -U postgres -h localhost` will prompt for the password you set, and the Python script will work with `host=localhost`, `user=postgres`, and your chosen password.

One caveat: some distros (notably certain Docker images or the official PostgreSQL apt repo) behave a little differently. The Docker `postgres` image, for instance, requires you to set `POSTGRES_PASSWORD` as an env var at container creation. But for a plain Linux package install, the answer is: user `postgres`, no password, peer auth.

How to change the password of the user postgres

Method 1: Using `psql` (most common)

Connect to PostgreSQL and run:

sql

```
ALTER USER postgres WITH PASSWORD 'new_password_here';
```

Or for any other user:

sql

```
ALTER USER username WITH PASSWORD 'new_password_here';
```

Method 2: Using `\password` in `psql` (more secure — doesn't show in logs)

bash

```
sudo -u postgres psql
```

Then inside `psql`:

```
\password postgres
```

It will prompt you to enter the new password twice without echoing it to the terminal or logs.

Method 3: From the command line (Linux)

bash

```
sudo -u postgres psql -c "ALTER USER postgres WITH PASSWORD 'new_password';"
```