# **Understanding the PostgreSQL Default User:**postgres**— Complete Guide for DBAs**

[](https://medium.com/@jramcloud1?source=post_page---byline--8d337b36c8b7---------------------------------------)

[Jeyaram Ayyalusamy](https://medium.com/@jramcloud1?source=post_page---byline--8d337b36c8b7---------------------------------------)

Following

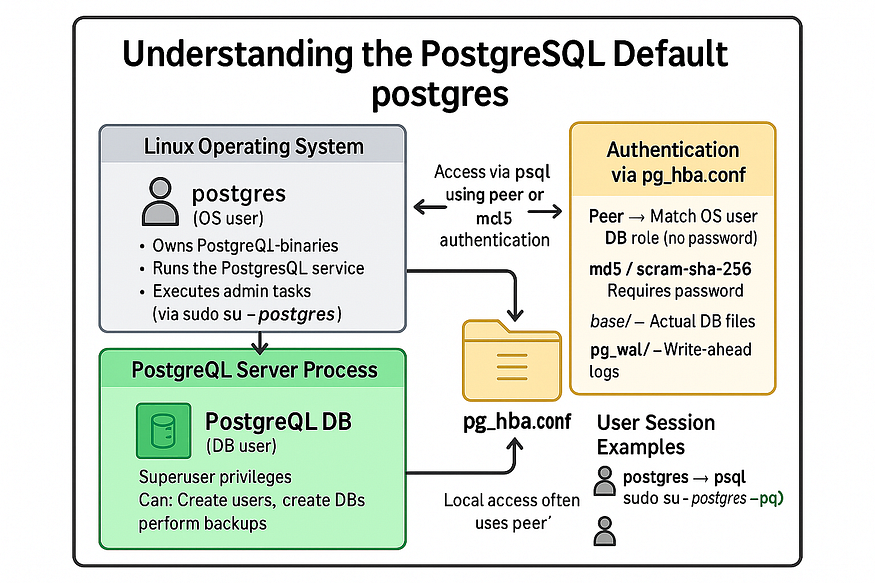
8 min read

·

Aug 1, 2025

1

Press enter or click to view image in full size



When you install ****PostgreSQL**** on any ****Linux server****, one of the first things you’ll encounter is the ****default system user: postgres****. This seemingly simple user plays a ****critical role**** in PostgreSQL's ecosystem — governing ****security, configuration, administration****, and more.

In this guide, we’ll dive deep into the ****purpose****, ****configuration****, ****authentication mechanisms****, and ****best practices**** surrounding the postgres user. Whether you're a ****beginner just getting started**** with PostgreSQL or an experienced admin managing ****large-scale production databases****, this article will help you ****confidently handle postgres access****.

## **⚙️ Why Is the**postgres**User Important?**

The postgres user has ****full privileges**** over the PostgreSQL server. It is used for:

* ****Starting and stopping**** the PostgreSQL service
* ****Creating and managing databases****
* ****Managing roles and permissions****
* ****Running administrative commands**** like psql, pg\_dump, vacuum, etc.
* Ensuring that ****PostgreSQL processes run securely**** under a non-root user

## **Verify the OS**postgres**user:**

cat /etc/passwd | grep -i postgres

✅ If PostgreSQL installed correctly, you’ll see an entry confirming the existence of this OS-level user.

## **🔐 Authentication Modes: How PostgreSQL Validates Users**

PostgreSQL controls user authentication through a configuration file called ****pg\_hba.conf**** (Host-Based Authentication). This file defines ****who can connect****, ****from where****, and ****how they are authenticated****.

## **📁 Step-by-Step: Locate and Understand**pg\_hba.conf

### **🔍 Step 1: Locate the File**

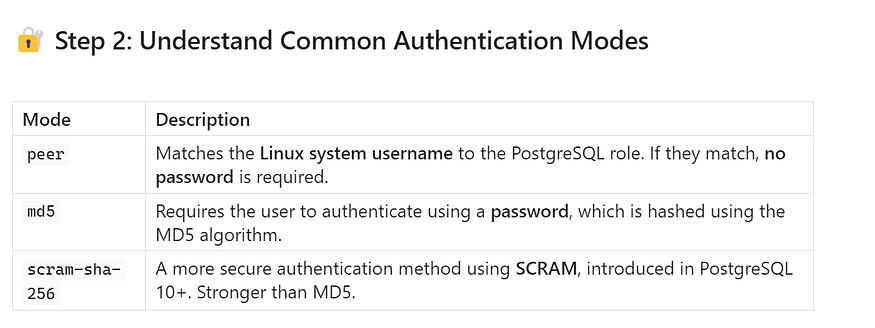
After installation, you can find the pg\_hba.conf file in the PostgreSQL data directory. For example, on version 17:

cat /var/lib/pgsql/17/data/pg\_hba.conf | grep peer

This file controls how users authenticate when connecting to the PostgreSQL server.

## **🔐 Step 2: Understand Common Authentication Modes**

Press enter or click to view image in full size



## **🧪 Step 3: Check Current Authentication Method**

Look inside the pg\_hba.conf file:

# TYPE DATABASE USER ADDRESS METHOD  
local all all peer

* In the above example, local connections to the postgres user are using ****peer authentication****.
* That means: if you’re logged into the Linux system as the postgres user, PostgreSQL will let you in ****without a password****, because the system username matches the database role.

## **⚙️ Optional: Change Authentication Mode (with Caution)**

If you want to change from peer to md5 or scram-sha-256:

1. Open the pg\_hba.conf file in an editor:

sudo vi /var/lib/pgsql/17/data/pg\_hba.conf

1. Modify the line:

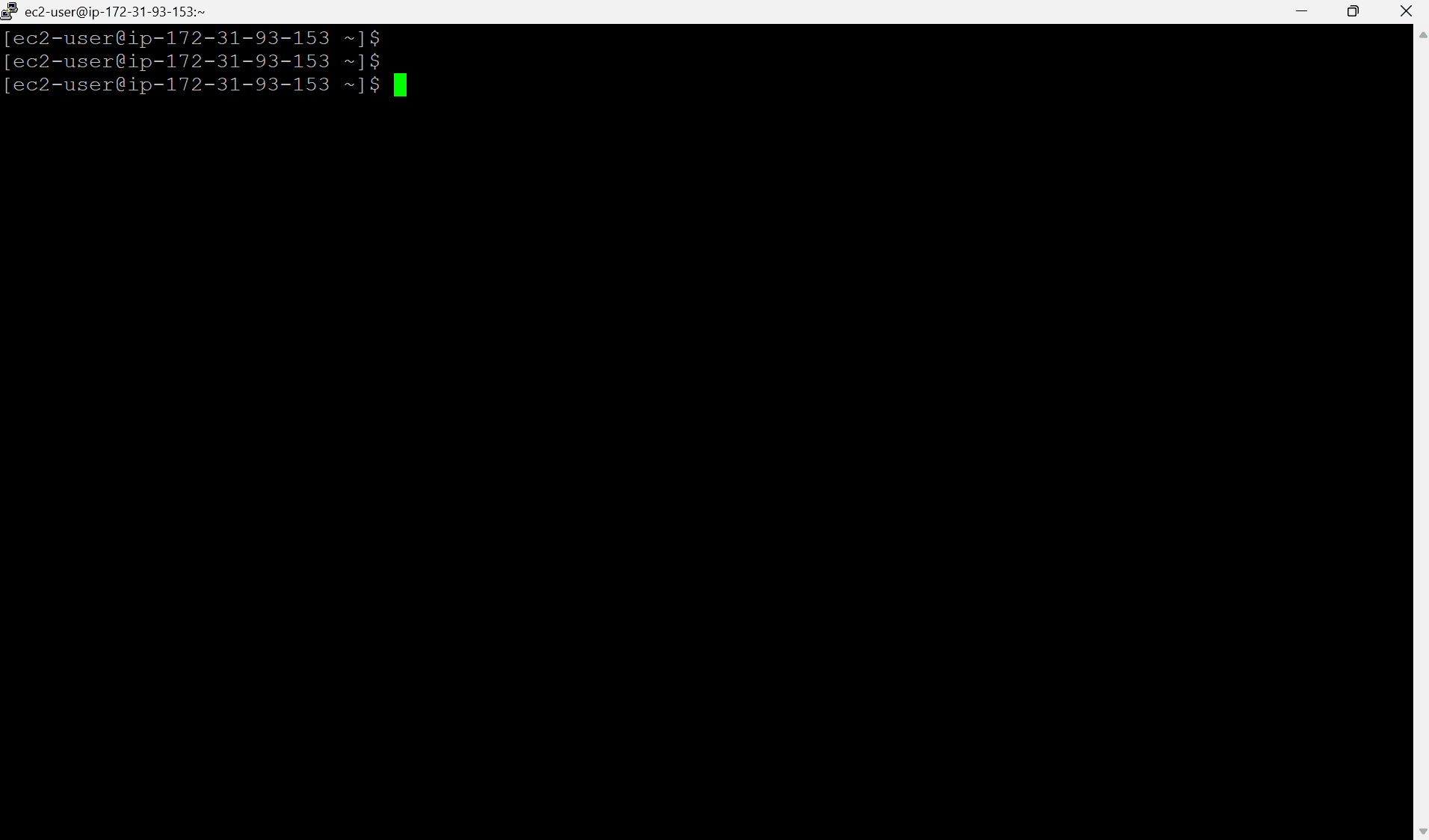
# From this: local all postgres peer # To this: local all postgres md5

1. Restart PostgreSQL to apply changes:

sudo systemctl restart postgresql-17

🔐 Always ensure the postgres user has a valid password set before changing from peer to md5 or scram.

Press enter or click to view image in full size



## **👤 Creating a New OS User and Setting It Up in PostgreSQL**

If you want to connect to PostgreSQL using a new Linux OS user (e.g., testuser), and use ****peer authentication**** or other methods securely, follow these ****step-by-step instructions****:

## **🧑‍💻 Step 1: Create a New Linux OS User**

Use the adduser or useradd command to create the user on your Linux system.

sudo adduser testuser

This will:

* Create a new home directory for testuser
* Set up login shell
* Prompt for password (if using adduser)

You can also use sudo useradd -m testuser if adduser is not available.

## **📦 Step 2: Switch to the New User**

Switch to the new user account:

sudo -i -u testuser

You’re now logged in as testuser — which is important for testing ****peer authentication**** later.

## **🗄️ Step 3: Create a Matching PostgreSQL Role**

Log in as the postgres system user and open the psql shell:

sudo -i -u postgres  
psql

Now create a PostgreSQL role named ****exactly the same**** as the OS user (testuser):

CREATE ROLE testuser LOGIN;

This creates a database user without a password (which works with peer authentication).

If you also want this user to ****create databases**** or do admin work:

ALTER ROLE testuser CREATEDB;

## **🧰 Step 4: Verify or Modify**pg\_hba.conf**for Peer Authentication**

Open the pg\_hba.conf file:

sudo vi /var/lib/pgsql/17/data/pg\_hba.conf

Ensure you have a line like this:

# TYPE DATABASE USER ADDRESS METHOD  
local all testuser peer

This tells PostgreSQL: ****if the Linux system user is testuser, allow them to log in to the testuser DB role without a password.****

Save the file and ****reload PostgreSQL****:

sudo systemctl reload postgresql-17

## **🧪 Step 5: Connect Using the New OS User**

Exit from postgres and switch to testuser:

exit  
sudo -i -u testuser

Now connect to PostgreSQL:

psql -d postgres

✅ If everything is configured correctly, this should log you in to PostgreSQL ****as the testuser role without needing a password****.

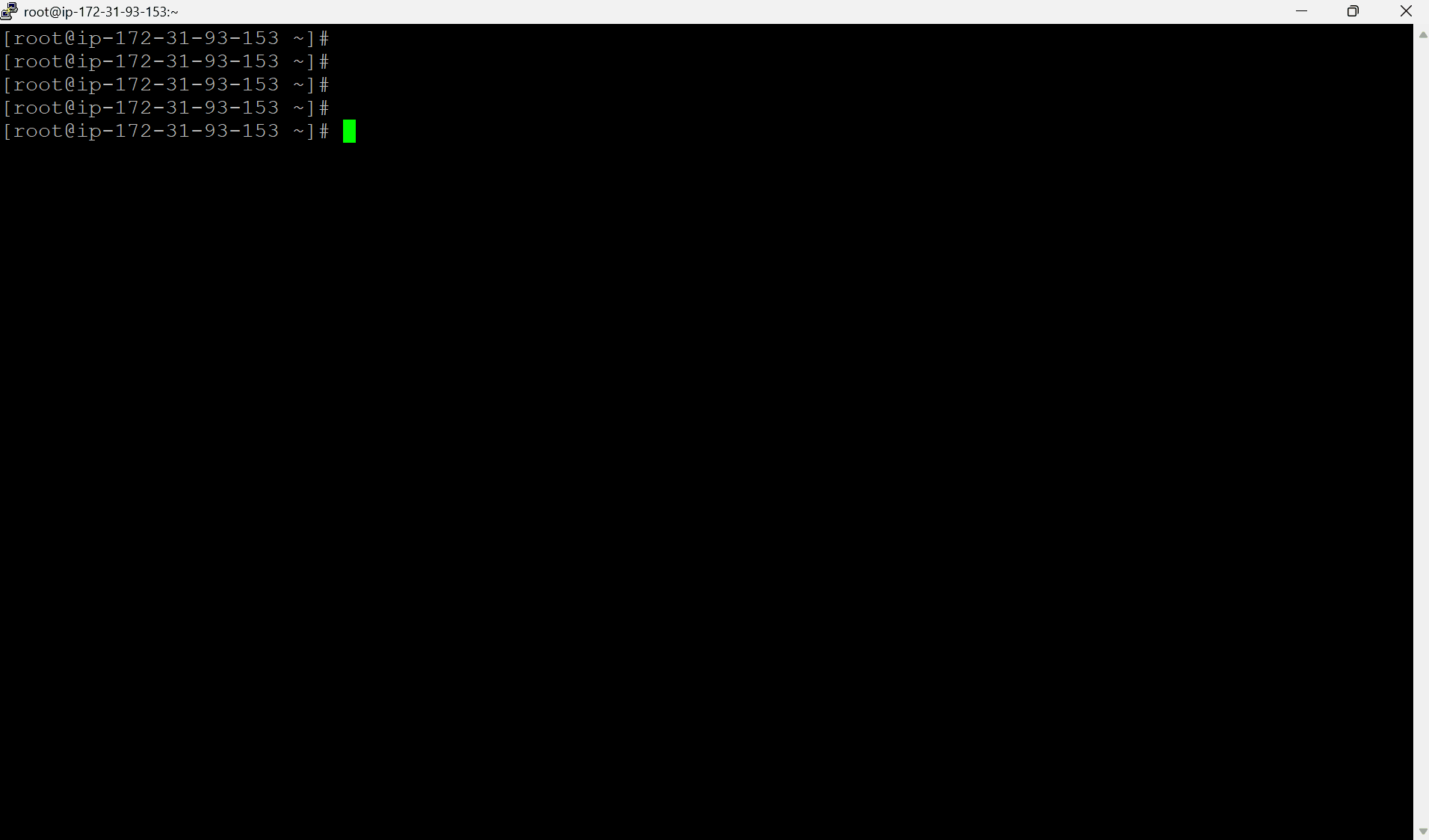
## **🎯 Optional: Use PGUSER Instead of Switching Users**

Instead of switching to testuser, you can connect as that user from any terminal by setting the PGUSER variable:

export PGUSER=testuser  
psql

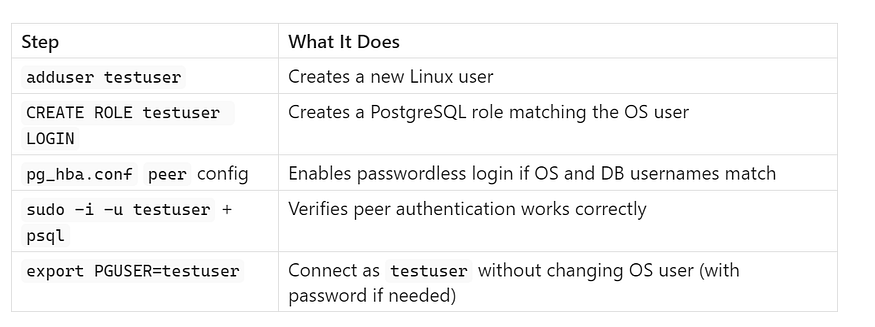
This is useful for scripting or managing connections from apps where switching OS users isn’t practical.

Press enter or click to view image in full size



## **✅ Final Recap: Full Flow Summary**

Press enter or click to view image in full size



This process demonstrates the ****tight integration between Linux and PostgreSQL**** authentication — a powerful feature that simplifies secure access control for local database users.

## **🛠️ Mastering PostgreSQL Administration: Access, Users, Roles, and Configuration**

When managing a PostgreSQL database on a Linux server, one of the first things you’ll need to do is ****access the server as the postgres user****, check and manage roles, and understand where configuration files are located.

In this detailed guide, we’ll cover:

1. How to access PostgreSQL using the postgres OS user
2. How to view existing users and their privileges
3. How to create and verify new users (roles)
4. How to locate the PostgreSQL data directory and key configuration files

Let’s walk through each step in detail.

## **🛠️ Accessing PostgreSQL as the**postgres**OS User**

To perform administrative tasks such as creating roles, databases, or editing configuration, you’ll typically need ****superuser access**** to PostgreSQL.

## **✅ Step-by-Step:**

1. ****Switch to the postgres system user:****

sudo su - postgres

1. ****Launch the PostgreSQL interactive terminal:****

psql

You’re now inside the psql shell ****as a superuser****, ready to execute commands with full privileges.

📌 You can confirm your identity inside psql by running:

SELECT current\_user;

You should see:

current\_user   
--------------  
 postgres

## **🔎 Checking Roles and User Privileges**

PostgreSQL uses a ****role-based access control system****. Roles can be users or groups, and they control who can access what.

## **✅ Step-by-Step: View All Roles (Users)**

Once inside psql as the postgres user:

1. ****List all roles using the meta-command:****

\du

This shows:

* Role names
* Attributes (e.g., Superuser, Create DB)
* Memberships

1. ****Alternatively, query system view:****

SELECT usename FROM pg\_user;

This returns just the list of usernames.

## **🔐 Step-by-Step: View Role Privileges**

To see what ****tables or schemas**** a specific user (like postgres) has access to:

SELECT table\_schema AS schema, table\_name AS table, privilege\_type AS privilege  
FROM information\_schema.table\_privileges  
WHERE grantee = 'postgres';

This will return a table like:

schema | table | privilege  
--------------------+---------------------------------------+------------  
 pg\_catalog | pg\_statistic | INSERT  
 pg\_catalog | pg\_statistic | SELECT  
 pg\_catalog | pg\_statistic | UPDATE  
 pg\_catalog | pg\_statistic | DELETE  
 pg\_catalog | pg\_statistic | TRUNCATE  
 pg\_catalog | pg\_statistic | REFERENCES  
 pg\_catalog | pg\_statistic | TRIGGER  
 pg\_catalog | pg\_type | INSERT  
 pg\_catalog | pg\_type | SELECT  
 pg\_catalog | pg\_type | UPDATE  
 pg\_catalog | pg\_type | DELETE  
 pg\_catalog | pg\_type | TRUNCATE  
 pg\_catalog | pg\_type | REFERENCES  
 pg\_catalog | pg\_type | TRIGGER  
 pg\_catalog | pg\_foreign\_table | INSERT  
 pg\_catalog | pg\_foreign\_table | SELECT  
 pg\_catalog | pg\_foreign\_table | UPDATE  
 pg\_catalog | pg\_foreign\_table | DELETE  
 pg\_catalog | pg\_foreign\_table | TRUNCATE  
 pg\_catalog | pg\_foreign\_table | REFERENCES  
 pg\_catalog | pg\_foreign\_table | TRIGGER  
 pg\_catalog | pg\_authid | INSERT  
 pg\_catalog | pg\_authid | SELECT  
 pg\_catalog | pg\_authid | UPDATE  
 pg\_catalog | pg\_authid | DELETE  
 pg\_catalog | pg\_authid | TRUNCATE  
 pg\_catalog | pg\_authid | REFERENCES  
 pg\_catalog | pg\_authid | TRIGGER  
 pg\_catalog | pg\_shadow | INSERT  
 pg\_catalog | pg\_shadow | SELECT  
 pg\_catalog | pg\_shadow | UPDATE  
 pg\_catalog | pg\_shadow | DELETE  
 pg\_catalog | pg\_shadow | TRUNCATE  
 pg\_catalog | pg\_shadow | REFERENCES  
 pg\_catalog | pg\_shadow | TRIGGER  
 pg\_catalog | pg\_roles | INSERT  
 pg\_catalog | pg\_roles | SELECT  
 pg\_catalog | pg\_roles | UPDATE  
 pg\_catalog | pg\_roles | DELETE  
 pg\_catalog | pg\_roles | TRUNCATE  
 pg\_catalog | pg\_roles | REFERENCES  
 pg\_catalog | pg\_roles | TRIGGER  
 pg\_catalog | pg\_statistic\_ext\_data | INSERT  
 pg\_catalog | pg\_statistic\_ext\_data | SELECT  
 pg\_catalog | pg\_statistic\_ext\_data | UPDATE  
 pg\_catalog | pg\_statistic\_ext\_data | DELETE  
 pg\_catalog | pg\_statistic\_ext\_data | TRUNCATE  
 pg\_catalog | pg\_statistic\_ext\_data | REFERENCES  
 pg\_catalog | pg\_statistic\_ext\_data | TRIGGER

🧠 Replace 'postgres' with any other role name to check their privileges.

## **👥 Creating and Managing New Users (Roles)**

PostgreSQL allows you to create new users (roles) using simple SQL commands.

## **✅ Step-by-Step: Create a New User**

1. Inside psql, run:

CREATE USER dbuser;

This creates a new user with ****no password and no privileges**** by default.

1. ****Verify that the user exists:****

\du

Or:

SELECT usename FROM pg\_user;

You should see dbuser listed.

## **❌ Check if the User Has Any Privileges**

Since the user is new and privileges haven’t been granted yet, check using:

SELECT table\_schema AS schema, table\_name AS table, privilege\_type AS privilege  
FROM information\_schema.table\_privileges  
WHERE grantee = 'dbuser';

Result:

(0 rows)

This confirms that testuser currently has no access to any tables.

✳️ Next steps would typically include granting privileges or assigning the user to a role.

## **📂 PostgreSQL Default Data Directory & Configuration Files**

Understanding where PostgreSQL stores its configuration and data files is crucial for backups, tuning, and troubleshooting.

## **✅ Step-by-Step: Find the Data Directory**

1. Log in to psql as the postgres user.
2. Run:

SHOW data\_directory;

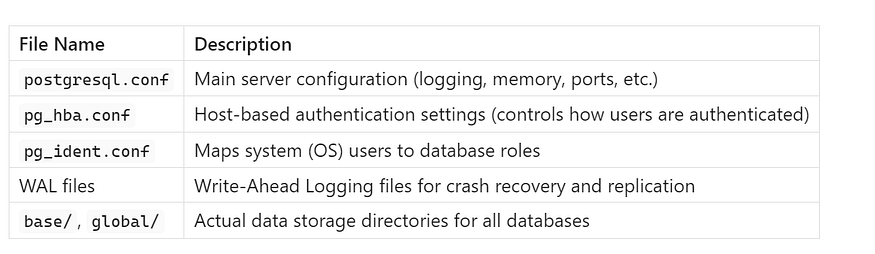
Typical output:

data\_directory   
----------------------------  
 /var/lib/pgsql/17/data

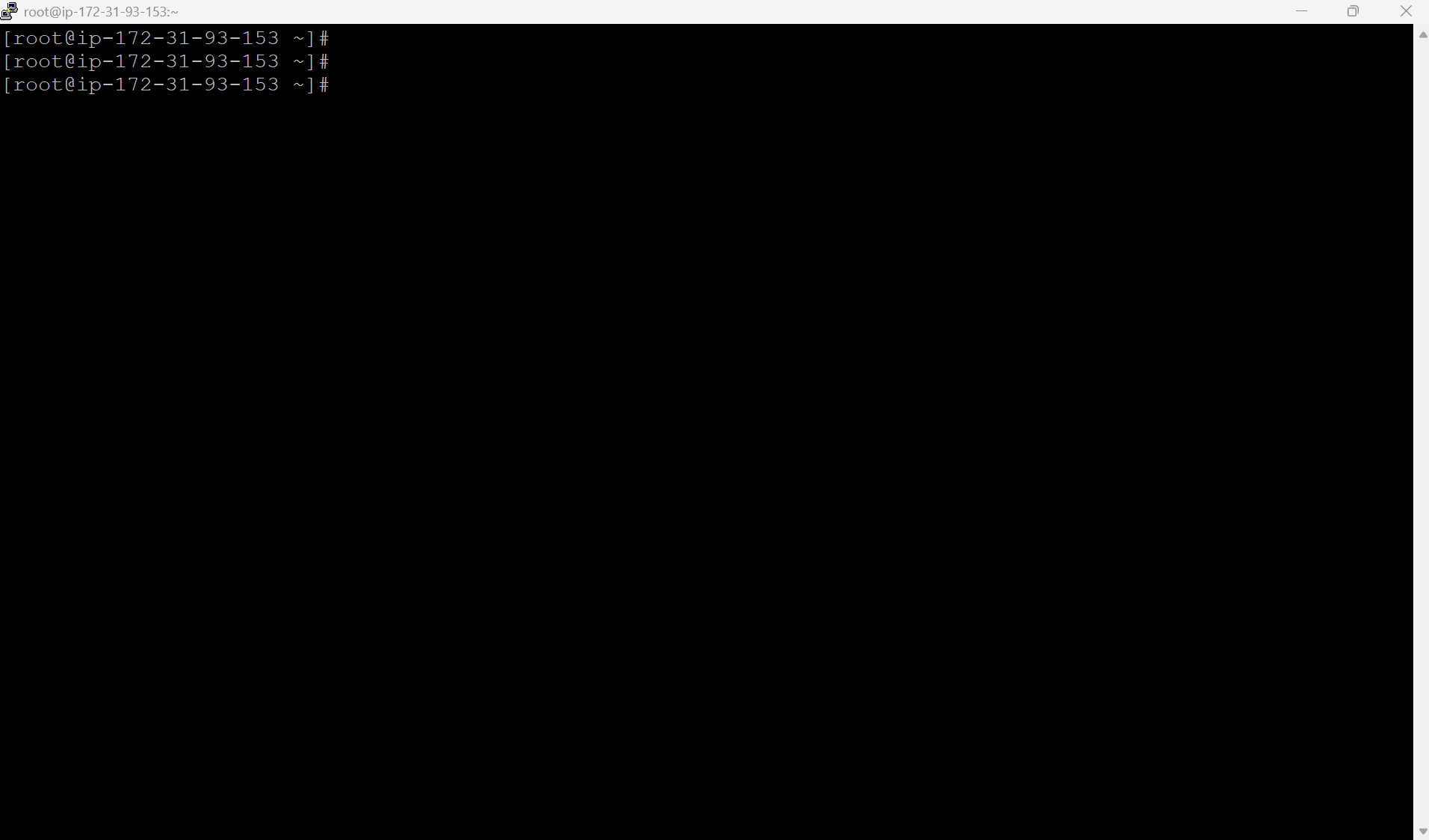
This is the ****PostgreSQL data directory**** — the heart of your PostgreSQL server.

## **📁 Key Files Inside the Data Directory**

Press enter or click to view image in full size



Press enter or click to view image in full size



## **✅ Conclusion**

* The postgres OS user is automatically created during installation and used for system-level management.
* The postgres PostgreSQL user is a ****superuser role**** with full control over the database.
* Peer authentication simplifies local admin tasks, while password-based authentication is better for remote access.
* New roles can be easily created and managed, offering granular control over database access.
* Understanding the postgres user helps maintain security, access control, and smooth PostgreSQL operations.

🔔 ****Pro Tip:****  
Always avoid using postgres for day-to-day application access.  
✅ Create application-specific users with only the necessary permissions.  
✅ Reserve postgres for administration and maintenance.