psql command line tutorial and cheat sheet

You've installed PostgreSQL. Now what? I assume you've been given a task that uses `psql` and you want to learn the absolute minimum to get the job done.

This is both a brief tutorial and a quick reference for the absolute least you need to know about `psql` . I assume you're familiar with the command line and have a rough idea about what database administration tasks, but aren't familiar with how to use `psql` to do the basics.

View on GitHub Pages or directly on GitHub

The PostgreSQL documentation is incredibly well written and thorough, but frankly, I didn't know where to start reading. This is my answer to that problem.

If you have any complaints or suggestions please let me know by sending your feedback to tomcampbell@gmail.com.

It shows how to do the following at the `psql` prompt:

If you don't have access to a live PostgreSQL installation at the moment we still have your back. You can follow through the examples and the output is shown as if you did type everything out.

The psql command line utility

Many administrative tasks can or should be done on your local machine, even though if database lives on the cloud. You can do some of them through a visual user interface, but that's not covered here. Knowing how to perform these operations on the command line means you can script them, and scripting means you can automate tests, check errors, and do data entry on the command line.

This section isn't a full cheat sheet for `psql` . It covers the most common operations and shows them roughly in sequence, as you'd use them in a typical work session.

What you need to know

Before using this section, you'll need:

- The user name and password for your PostgreSQL database

- The IP address of your remote instance

Command-line prompts on the operating system

The `$` starting a command line in the examples below represents your operating system prompt. Prompts are configurable so it may well not look like this. On Windows it might look like `C:\Program Files\PostgreSQL>` but Windows prompts are also configurable.

```
$ psql -U sampleuser -h localhost
```

A line starting with `#` represents a comment. Same for everything to the right of a `#`. If you accidentally type it or copy and paste it in, don't worry. Nothing will happen.

```
This worked to connect to Postgres on DigitalOcean
# -U is the username (it will appear in the \l command)
# -h is the name of the machine where the server is running.
# -p is the port where the database listens to connections. Default is 5432.
# -d is the name of the database to connect to. I think DO generated this for me, or maybe PostgreSQL.
# Password when asked is csizllepewdypieiib
$ psql -U doadmin -h production-sfo-test1-do-user-4866002-0.db.ondigitalocean.com -p 25060 -d mydb

# Open a database in a remote location.
$ psql -U sampleuser -h production-sfo-test1-do-user-4866002-0.db.ondigitalocean.com -p 21334
```

Using psql

You'll use `psql` (aka the PostgreSQL interactive terminal) most of all because it's used to create databases and tables, show information about tables, and even to enter information (records) into the database.

Quitting pqsql

Before we learn anything else, here's how to quit `psql` and return to the operating system prompt. You type backslash, the letter `q`, and then you press the Enter or return key.

```
# Press enter after typing \q
# Remember this is backslash, not forward slash
postgres=# \q
```

This takes you back out to the operating system prompt.

Opening a connection locally

A common case during development is opening a connection to a local database (one on your own machine). Run `psql` with `-U` (for user name) followed by the name of the database, `postgres` in this example:

```
# Log into Postgres as the user named postgres
$ psql -U postgres
```

Opening a connection remotely

To connect your remote PostgreSQL instance from your local machine, use `psql` at your operating system command line. Here's a typical connection.

```
# -U is the username (it will appear in the \l command)
# -h is the name of the machine where the server is running.
# -p is the port where the database listens to connections. Default is 5432.
# -d is the name of the database to connect to. I think DO generated this for me, or maybe PostgreSQL.
$ psql -U doadmin -h production-sfo-test1-do-user-4866002-0.db.ondigitalocean.com -p 25060 -d defaultdb
```

Here you'd enter the password. In case someone is peering over your shoulder, the characters are hidden. After you've entered your information properly you'll get this message (truncated for clarity):

Looking at the psql prompt

A few things appear, then the `psql` prompt is displayed. The name of the current database appears before the prompt.

```
psql (11.1, server 11.0)
Type "help" for help.

postgres=#
```

At this point you're expected to type commands and parameters into the command line.

psql vs SQL commands

`psql` has two different kinds of commands. Those starting with a backslash are for `psql` itself, as illustrated by the use of `\q` to quit.

Those starting with valid SQL are of course interactive SQL used to create and modify PostgreSQL databases.

Warning: SQL commands end with a semicolon!

One gotcha is that almost all SQL commands you enter into `psql` must end in a semicolon.

- For example,suppose you want to remove a table named `sample_property_5`. You'd enter this command:

```
postgres=# DROP TABLE "sample_property_5";
```

It's easy to forget. If you do forget the semicolon, you'll see this perplexing prompt. Note that a `[` has been inserted before the username portion of the prompt, and another prompt appears below it:

```
[postgres=# DROP TABLE "sample_property_5"
postgres=#
```

When you do, just remember to finish it off with that semicolon:

```
[postgres=# DROP TABLE "sample_property_5"
postgres=# ;
```

Scrolling through the command history

- Use the up and down arrow keys to move backwards and forwards through the command history.

Getting information about databases

These aren't SQL commands so just press Enter after them. Remember that:

- When there's more output than fits the screen, it pauses. Press space to continue
- If you want to halt the output, press `q` .

\h Help

```
# Get help. Note it's a backslash, not a forward slash.
postgres=# \h
```

You'll get a long list of commands, then output is paused:

```
Available help:
  ABORT                              CREATE USER
  ...
  ALTER AGGREGATE                    CREATE USER MAPPING
  ALTER PROCEDURE                    DROP INDEX
:
```

- Press space to continue, or `q` to stop the output.

You can get help on a particular item by listing it after the `\h` command.

- For example, to get help on `DROP TABLE` :

```
postgres=# \h drop table
```

You'll get help on just that item:

```
Command:     DROP TABLE
Description: remove a table
Syntax:
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

\l List databases

What most people think of as a database (say, a list of customers) is actually a table. A database is a set of tables, information about those tables, information about users and their permissions, and much more. Some of these databases (and the tables within) are updated automatically by PostgreSQL as you use them.

To get a list of all databases:

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 visitor   | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 markets   | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 tom       | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

You can get info on a single database by following the `\l` prompt with its name.

- For example, to view information about the `template0` database:

```
postgres=# \l template0
```

The output would be:

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
```

\l+ List databases with size, tablespace, and description

To get additional information on the space consumed by database tables and comments describing those tables, use `\l+` :

```
postgres=# \l+
```

\x Expand/narrow table lists

Use `\x` (X for eXpanded listing) to control whether table listings use a wide or narrow format.

Command	Effect
`\x off`	Show table listings in wide format
`\x on`	Show table listings in narrow format
`\x`	Reverse the previous state
`\x auto`	Use terminal to determine format

Example: Here's an expanded listing:

```
/* List all databases. */
postgres=# \l

                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 foo       | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 foobarino | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 tom       | tom      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(6 rows)
```

Use `\x on` for narrower listings:

```
/* Turn on narrow listings. */
postgres=# \x on
postgres=# \l


-[ RECORD 1 ]-----+----------------------
Name              | foo
Owner             | tom
Encoding          | UTF8
```

```
Collate           | en_US.UTF-8
Ctype             | en_US.UTF-8
Access privileges |
-[ RECORD 2 ]-----+---------------------
Name              | foobarino
Owner             | tom
Encoding          | UTF8
Collate           | en_US.UTF-8
Ctype             | en_US.UTF-8
Access privileges |
-[ RECORD 3 ]-----+---------------------
Name              | postgres
Owner             | postgres
Encoding          | UTF8
Collate           | en_US.UTF-8
Ctype             | en_US.UTF-8
Access privileges |
```

\c Connect to a database

To see what's inside a database, connect to it using `\c` followed by the database name. The prompt changes to match the name of the database you're connecting to. (The one named `postgres` is always interesting.) Here we're connecting to the one named `markets` :

```
postgres=# \c markets
psql (11.1, server 11.0)
You are now connected to database "markets" as user "tom".
markets=#
```

\dt Display tables

- Use `\dt` to list all the tables (technically, *relations*) in the database:

```
markets=# \dt

                    List of relations
 Schema |             Name             | Type  |  Owner
--------+------------------------------+-------+----------
 public | addresspool                  | table | tom
 public | adlookup                     | table | tom
 public | bidactivitysummary           | table | tom
 public | bidactivitydetail            | table | tom
 public | customerpaymentsummary       | table | tom
 ...
```

- If you choose a database such as `postgres` there could be many tables. Remember you can pause output by pressing space or halt it by pressing `q` .

\d and \d+ Display columns (field names) of a table

To view the schema of a table, use `\d` followed by the name of the table.

- To view the schema of a table named `customerpaymentsummary` , enter

```
markets=# \d customerpaymentsummary

                            Table "public.customerpaymentsummary"
            Column            |            Type             | Collation | Nullable | Default
------------------------------+-----------------------------+-----------+----------+--------
 usersysid                    | integer                     |           | not null |
 paymentattemptsall           | integer                     |           |          |
 paymentattemptsmailin        | integer                     |           |          |
 paymentattemptspaypal        | integer                     |           |          |
 paymentattemptscreditcard    | integer                     |           |          |
 paymentacceptedoutagecredit  | integer                     |           |          |
 totalmoneyin                 | numeric(12,2)               |           |          |
 updatewhen1                  | timestamp without time zone |           |          |
 updatewhen2                  | timestamp without time zone |           |          |
```

To view more detailed information on a table, use `\d+` :

```
markets=# \d customerpaymentsummary

                            Table "public.customerpaymentsummary"
            Column            |            Type             | Collation | Nullable | Default | Storage | Stats target
------------------------------+-----------------------------+-----------+----------+---------+---------+-------------
 usersysid                    | integer                     |           | not null |         | plain   |
 paymentattemptsall           | integer                     |           |          |         | plain   |
 paymentattemptsmailin        | integer                     |           |          |         | plain   |
 paymentattemptspaypal        | integer                     |           |          |         | plain   |
 paymentattemptscreditcard    | integer                     |           |          |         | plain   |
 paymentacceptedoutagecredit  | integer                     |           |          |         | plain   |
 totalmoneyin                 | numeric(12,2)               |           |          |         | main    |
 updatewhen1                  | timestamp without time zone |           |          |         | plain   |
 updatewhen2                  | timestamp without time zone |           |          |         | plain   |

Indexes:
```

\du Display user roles

- To view all users and their roles, use `\du` :

```
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 smanager   | Superuser                                                  | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 tom        | Superuser, Create role, Create DB                          | {}
```

- To view the role of a specific user, pass it after the `\du` command. For example, to see the only `tom` role:

```
postgres=# \du tom
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 tom        | Superuser, Create role, Create DB                          | {}
```

Creating a database

Before you add tables, you need to create a database to contain those tables. That's not done with `psql`, but instead it's done with `createdb` (a separate external command; see the PostgreSQL createdb documentation) at the operating system command line:

```
# Replace markets with your database name
$ createdb markets
```

On success, there is no visual feedback. Thanks, PostgreSQL.

Adding tables and records

Creating a table (CREATE TABLE)

To add a table schema to the database:

```
postgres=# create table if not exists product (
  id               SERIAL,
  name             VARCHAR(100) NOT NULL,
  sku              CHAR(8)
);
```

And `psql` responds with:

```
CREATE TABLE
```

For more see `CREATE TABLE` in the PostgreSQL official docs.

Adding a record (INSERT INTO)

- Here's how to add a record, populating every field:

```
# The id field is an automatically assigned
# when you use DEFAULT. The serial primary key means
# that number will be increased by at least
# 1 and assigned to that same field when
# a new record is created.
# Using DEFAULT is a placeholder.
# In its place PostgreSQL automatically generates a unique integer for it.
postgres=# INSERT INTO product VALUES(DEFAULT, 'Apple, Fuji', '4131');
```

PostgreSQL responds with:

```
INSERT 0 1
```

- Try it again and you get a simliar response.

```
postgres=# INSERT INTO product VALUES(DEFAULT, 'Banana', '4011');
INSERT 0 1
```

Adding (inserting) several records at once

- You can enter a list of records using this syntax:

```
postgres=# INSERT INTO product VALUES
(DEFAULT, 'Carrots', 4562),
(DEFAULT, 'Durian', 5228)
;
```

Adding only specific (columns) fields from a record

You can add records but specify only selected fields (also known as columns). MySQL will use common sense default values for the rest.

In this example, only the `name` field will be populated. The `sku` column is left blank, and the `id` column is incremented and inserted.

Two records are added:

```
postgres=# INSERT INTO product (name) VALUES
('Endive'),
('Figs')
;
```

PostgreSQL responds with the number of records inserted:

```
INSERT 0 2
```

For more on INSERT, see `INSERT` in the PostgreSQL official docs

Doing a simple query–get a list of records (SELECT)

Probably the most common thing you'll do with a table is to obtain information about it with the `SELECT` statement. It's a huge topic

- Let's list all the records in the `product` table:

```
postgres=# SELECT * FROM product;
```

The response:

```
postgres=# select * from product;
 id |     name      |   sku
----+-------------+----------
  1 | Apple, Fuji | 4131
  2 | Banana      | 4011
(2 rows)
```

Note

If your table has mixed case objects such as column names or indexes, you'll need to enclose them in double quotes. For example, If a column name were `Product` instead of `product` your query would need to look like this:

```
SELECT * FROM "product";
```

For more on SELECT, see the `SELECT` in the PostgreSQL official docs.

Maintenance and operations issues

Timing

\t Timing SQL operations

Use `\t` to show timing for all SQL operations performed.

Command	Effect
`\timing off`	Disable timing of SQL operations
`\timing on`	Show timing after all SQL operations
`\timing`	Toggle (reverse) the setting

Example of \t Timing command

```
tom=# insert into todo values ('Retry on Android before app submission,'8.x and earlier');
INSERT 0 1
tom=# \timing on
Timing is on.
tom=# insert into todo values ('Correct footer bug','Mobile version only');
INSERT 0 1
Time: 1.067 ms
tom=# insert into todo values ('Retry on Android before app submission', '8.x and earlier');
INSERT 0 1
Time: 23.312 ms
tom=# \timing
Timing is off.
```

Watch

The `\watch` command repeats the previous command at the specified interval. To use it, enter the SQL command you want repeated, then use `\watch` followed by the number of seconds you want for the interval between repeats, for rexample, `\watch 1` to repeat it every second.

Example of the \Watch command

Here's an example of using `\watch` to see if any records have been inserted within the last 5 seconds.

```
tom=# select count(*);
  count
--------
    726
(726 rows)

tom=# \watch 5
Mon Nov 16 13:50:36 2020 (every 2s)

  count
--------
    726
(726 rows)
```

```
Mon Nov 16 13:50:38 2020 (every 2s)

  count
--------
    726
(726 rows)

Mon Nov 16 13:50:40 2020 (every 2s)

  count
--------
    726
(726 rows)
```

Locate the pg_hba.conf file

Postgres configuration is stored in a file named `pg_hba.conf` *somewhere* in the file system, but that location varies widely. The way to find it is to use `show hba_file` like this:

```
show  hba_file;
```

See below for hot reloading this file while Postgres is running.

Reload the configuration file while Postgres is running

If you make changes to the `pg_hba.conf` Postgres configuration sometimes you need to restart. But you may just choose to reload the `pg_hba.conf` configuration file like this:

```
SELECT pg_reload_conf();
```

Reference

- PostgreSQL offical docs: Server Administration
- `psql` , a.k.a the PostgreSQL interactive terminal
- `createdb` in the PostgreSQL offical docs
- `CREATE TABLE` in the PostgreSQL official docs
- `INSERT` in the PostgreSQL official docs