

# Базовый инструментарий Использование psql



## Авторские права

© Postgres Professional, 2017–2024

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Алексей Береснев

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

## Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

## Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

## Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Запуск psql и подключение к БД

Получение справочной информации

Работа в psql

Настройка

Терминальный клиент для работы с PostgreSQL

Поставляется вместе с СУБД

Используется администраторами и разработчиками для интерактивной работы и выполнения скриптов

Для работы с СУБД PostgreSQL существуют различные сторонние инструменты, рассмотрение которых не входит в рамки курса.

В курсе мы будем использовать терминальный клиент `psql`. Этот клиент может показаться непривычным для тех, кто предпочитает графические инструменты; тем не менее, он весьма удобен.

Это единственный клиент, поставляемый вместе с СУБД. Навыки работы с `psql` пригодятся разработчикам и администраторам БД вне зависимости от того, с каким инструментом они будут работать дальше.

<https://postgrespro.ru/docs/postgresql/16/app-psql>

## Запуск

```
$ psql -d база -U пользователь -h узел -p порт
```

## Новое подключение в psql

```
=> \connect база пользователь узел порт
```

## Информация о текущем подключении

```
=> \conninfo
```

При запуске psql определяет параметры подключения: имя базы данных, имя пользователя, имя сервера, номер порта. Если эти параметры не указаны, psql попытается подключиться, используя значения по умолчанию:

- *база* — совпадает с именем пользователя;
- *пользователь* — совпадает с именем пользователя ОС;
- *узел* — соединение через Unix-socket;
- *порт* — обычно 5432.

Если требуется выполнить новое подключение не выходя из psql, нужно выполнить команду `\connect`. Значения по умолчанию для нее берутся из текущего соединения.

Команда `\conninfo` выдает информацию о текущем подключении.

Дополнительная информация о возможностях настройки подключения:

<https://postgrespro.ru/docs/postgresql/16/libpq-envvars>

<https://postgrespro.ru/docs/postgresql/16/libpq-pgservice>

<https://postgrespro.ru/docs/postgresql/16/libpq-pgpass>

## В командной строке ОС

```
$ psql --help  
$ man psql
```

## В psql

=> \?	список команд psql
=> \? variables	переменные psql
=> \h[elp]	список команд SQL
=> \h команда	синтаксис команды SQL
=> \q	выход

Справочную информацию по psql можно получить не только в документации, но и прямо в системе.

psql с ключом `--help` выдает справку по запуску. А если в системе была установлена документация, то справочное руководство можно получить командой `man psql`.

psql умеет выполнять команды SQL и свои собственные команды. Все команды psql начинаются с обратной косой черты и, как правило, их можно сокращать до первой буквы.

Внутри psql есть возможность получить список и краткое описание команд psql: `\?`.

Команда `\help` выдает список команд SQL, которые поддерживает сервер, а также синтаксис конкретной команды SQL.

И еще одна команда, которую полезно знать, хоть она и не имеет отношения к справке: это `\q` — выход из psql. В качестве альтернативы для выхода можно использовать команды `exit` и `quit`.

## Выполнение команд SQL и форматирование результатов

Запускаем psql:

```
student$ psql
```

Проверим текущее подключение:

```
=> \conninfo
```

You are connected to database "student" as user "student" via socket in  
"/var/run/postgresql" at port "5432".

С параметрами по умолчанию мы подключились к базе данных student под пользователем student. Подробнее о базах данных и пользователях будет рассказано в следующих темах курса.

Команда \connect выполняет новое подключение, не покидая psql.

Команды SQL, в отличие от команд psql, могут располагаться на нескольких строках. Для отправки команды SQL на выполнение завершаем ее точкой с запятой:

```
=> SELECT schemaname, tablename, tableowner
FROM pg_tables
LIMIT 5;
```

schemaname	tablename	tableowner
pg_catalog	pg_statistic	postgres
pg_catalog	pg_type	postgres
pg_catalog	pg_foreign_table	postgres
pg_catalog	pg_authid	postgres
pg_catalog	pg_statistic_ext_data	postgres

(5 rows)

Утилита psql умеет выводить результат запросов в разных форматах. Вот только некоторые из них:

- формат с выравниванием значений;
- формат без выравнивания;
- расширенный формат.

Формат с выравниванием используется по умолчанию. Ширина столбцов выровнена по значениям. Также выводится строка заголовков и итоговая строка.

Команды psql для переключения режима выравнивания:

- \a — переключает режим с выравниванием и без выравнивания;
- \t — переключает отображения строки заголовка и итоговой строки.

Отключим выравнивание, вывод заголовка и итоговой строки, а в качестве разделителя столбцов установим пробел:

```
=> \t \a
```

Tuples only is on.  
Output format is unaligned.

```
=> \pset fieldsep ' '
```

Field separator is " ".

```
=> SELECT schemaname, tablename, tableowner FROM pg_tables LIMIT 5;
```

```
pg_catalog pg_statistic postgres
pg_catalog pg_type postgres
pg_catalog pg_foreign_table postgres
pg_catalog pg_authid postgres
pg_catalog pg_statistic_ext_data postgres
```

```
=> \t \a
```

Tuples only is off.  
Output format is aligned.

Расширенный формат удобен, когда нужно вывести много столбцов для одной или нескольких записей:

```
=> \x
```

Expanded display is on.

```
=> SELECT * FROM pg_tables WHERE tablename = 'pg_class';
```

```
-[ RECORD 1 ]-----
schemaname | pg_catalog
tablename  | pg_class
tableowner | postgres
tablespace |
hasindexes | t
hasrules   | f
hastriggers| f
rowsecurity| f
```

```
=> \x
```

Expanded display is off.

Расширенный формат можно установить только для одного запроса, если в конце вместо точки с запятой указать \gx:

```
=> SELECT * FROM pg_tables WHERE tablename = 'pg_proc' \gx
```

```
-[ RECORD 1 ]-----
schemaname | pg_catalog
tablename  | pg_proc
tableowner | postgres
tablespace |
hasindexes | t
hasrules   | f
hastriggers | f
rowsecurity | f
```

Все возможности форматирования результатов запросов доступны через команду \pset. А без параметров она покажет текущие настройки форматирования:

```
=> \pset
```

```
border          1
columns         0
csv_fieldsep    ','
expanded        off
fieldsep        ' '
fieldsep_zero   off
footer          on
format          aligned
linestyle       ascii
null            ''
numericlocale   off
pager           1
pager_min_lines 0
recordsep       '\n'
recordsep_zero  off
tableattr
title
tuples_only     off
unicode_border_linestyle single
unicode_column_linestyle single
unicode_header_linestyle single
xheader_width   full
```

---

## Взаимодействие с ОС

В psql можно выполнять команды shell:

```
=> \! pwd
```

```
/home/student
```

Можно установить переменную окружения операционной системы:

```
=> \setenv HELLO Hello
```

```
=> \! echo $HELLO
```

```
Hello
```

Можно записать вывод команды в файл с помощью \o[ut]:

```
=> \o tmp/dba1_log
```

```
=> SELECT schemaname, tablename, tableowner FROM pg_tables LIMIT 5;
```

На экран ничего не попало. Посмотрим в файле:

```
=> \! cat tmp/dba1_log
```

```
schemaname |      tablename      | tableowner
-----+-----+-----
pg_catalog | pg_statistic         | postgres
pg_catalog | pg_type              | postgres
pg_catalog | pg_foreign_table     | postgres
pg_catalog | pg_authid            | postgres
pg_catalog | pg_statistic_ext_data | postgres
(5 rows)
```

Вернем вывод на экран:

```
=> \o
```

---

## Выполнение скриптов

Еще один вариант отправить запрос на выполнение — команда \g. В скобках можно указать параметры форматирования только для этого запроса.

Вывод запроса можно направить команде ОС, если указать ее после вертикальной черты. Например, можно вывести результат запроса на экран, пронумеровав строки:

```
=> SELECT format('SELECT count(*) FROM %I;', tablename)
FROM pg_tables
LIMIT 3
\g (tuples_only=on format=unaligned) | cat -n

 1 SELECT count(*) FROM pg_statistic;
 2 SELECT count(*) FROM pg_type;
 3 SELECT count(*) FROM pg_foreign_table;
```

В команде \g можно указать имя файла, в который будет направлен вывод:

```
=> SELECT format('SELECT count(*) FROM %I;', tablename)
FROM pg_tables
LIMIT 3
\g (tuples_only=on format=unaligned) tmp/dbal_log
```

Вот что получилось в файле:

```
=> \! cat tmp/dbal_log

SELECT count(*) FROM pg_statistic;
SELECT count(*) FROM pg_type;
SELECT count(*) FROM pg_foreign_table;
```

---

Выполняем теперь этот файл как скрипт с помощью \i[include]:

```
=> \i tmp/dbal_log

count
-----
  409
(1 row)

count
-----
  613
(1 row)

count
-----
    0
(1 row)
```

Другие способы выполнить команды из файла:

- psql < filename
- psql -f filename

---

В предыдущем примере можно обойтись и без создания файла, если завершить запрос командой \gexec:

```
=> SELECT format('SELECT count(*) FROM %I;', tablename)
FROM pg_tables
LIMIT 3
\gexec

count
-----
  409
(1 row)

count
-----
  613
(1 row)

count
-----
    0
(1 row)
```

---

Команда gexec считает, что в каждом столбце каждой строки выборки содержится SQL-оператор, и выполняет эти операторы один за другим.

## Переменные psql и управляющие конструкции

По аналогии с shell, psql имеет собственные переменные, среди которых есть ряд встроенных (имеющих особый смысл для psql).

Запомним в переменной psql User значение переменной окружения USER:

```
=> \getenv User USER
```

И установим переменную Test:

```
=> \set Test Hi
```

Чтобы подставить ее значение, надо предварить имя переменной двоеточием:

```
=> \echo :Test :User!
```

Hi student!

Сбросить переменную можно так:

```
=> \unset Test
```

```
=> \echo :Test
```

:Test

---

Результат запроса можно записать в переменную. Для этого запрос нужно завершить командой \gset:

```
=> SELECT now() AS curr_time \gset
```

```
=> \echo :curr_time
```

2025-06-23 23:21:34.462306+03



Запрос должен возвращать только одну запись.

Без параметров \set выдает значения всех установленных переменных:

```
=> \set

AUTOCOMMIT = 'on'
COMP_KEYWORD_CASE = 'preserve-upper'
DBNAME = 'student'
ECHO = 'none'
ECHO_HIDDEN = 'off'
ENCODING = 'UTF8'
ERROR = 'false'
FETCH_COUNT = '0'
HIDE_TABLEAM = 'off'
HIDE_TOAST_COMPRESSION = 'off'
HISTCONTROL = 'none'
HISTFILE = 'hist'
HISTSIZE = '500'
HOST = '/var/run/postgresql'
IGNOREEOF = '0'
LAST_ERROR_MESSAGE = ''
LAST_ERROR_SQLSTATE = '00000'
ON_ERROR_ROLLBACK = 'off'
ON_ERROR_STOP = 'off'
PORT = '5432'
PROMPT1 = '%R%x%# '
PROMPT2 = '%R%x%# '
PROMPT3 = '>> '
QUIET = 'off'
ROW_COUNT = '1'
SERVER_VERSION_NAME = '16.9 (Ubuntu 16.9-1.pgdg24.04+1)'
SERVER_VERSION_NUM = '160009'
SHELL_ERROR = 'false'
SHELL_EXIT_CODE = '0'
SHOW_ALL_RESULTS = 'on'
SHOW_CONTEXT = 'errors'
SINGLELINE = 'off'
SINGLESTEP = 'off'
SQLSTATE = '00000'
USER = 'student'
User = 'student'
VERBOSITY = 'default'
VERSION = 'PostgreSQL 16.9 (Ubuntu 16.9-1.pgdg24.04+1) on x86_64-pc-linux-gnu, compiled
by gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0, 64-bit'
VERSION_NAME = '16.9 (Ubuntu 16.9-1.pgdg24.04+1)'
VERSION_NUM = '160009'
curr_time = '2025-06-23 23:21:34.462306+03'
```

В скриптах можно использовать условные операторы.

Предположим, что мы хотим проверить, установлено ли значение переменной `working_dir`, и если нет, то присвоить ей имя текущего каталога. Для проверки существования переменной можно использовать следующую конструкцию, возвращающую логическое значение:

```
=> \echo :{?working_dir}

FALSE

Следующий условный оператор psql проверяет существование переменной и при необходимости устанавливает значение по умолчанию:

=> \if :{?working_dir}
-- переменная определена
\else
-- в качестве значения можно установить результат выполнения команды 0C
\set working_dir `pwd`
\endif
```

Теперь мы можем быть уверены, что переменная `working_dir` определена:

```
=> \echo :working_dir

/home/student
```

### Команды для работы с системным каталогом

С помощью серии команд, в основном начинающихся на `\d`, можно быстро и удобно получать информацию об объектах БД.

Например:

```
=> \d pg_tables

          View "pg_catalog.pg_tables"
  Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----
 schemaname | name |          |          |
 tablename | name |          |          |
 tableowner | name |          |          |
 tablespace | name |          |          |
 hasindexes | boolean |          |          |
 hasrules | boolean |          |          |
 hastriggers | boolean |          |          |
 rowsecurity | boolean |          |          |
```

Подробнее такие команды будут рассмотрены позже.

# Настройка psql

При запуске psql выполняются два скрипта (при их наличии):

- сначала общий системный скрипт psqlrc;
- затем пользовательский файл .psqlrc.

Пользовательский файл должен располагаться в домашнем каталоге, а расположение системного скрипта можно узнать командой:

```
student$ pg_config --sysconfdir
/etc/postgresql-common
```

По умолчанию оба файла отсутствуют.

В эти файлы можно поместить команды для настройки сеанса, например:

- приглашение psql;
- программу постраничного просмотра результатов запросов;
- переменные для хранения текста часто используемых команд.

Для примера запишем в переменную top5 текст запроса на получение пяти самых больших по размеру таблиц:

```
=> \set top5 'SELECT tablename, pg_total_relation_size(schemaname||'.'.')||tablename) AS bytes FROM pg_tables ORDER BY bytes DESC LIMIT 5;'
```

Для выполнения запроса достаточно набрать:

```
=> :top5

  tablename  | bytes
-----+-----
pg_proc      | 1245184
pg_rewrite   |  745472
pg_attribute |  720896
pg_description | 630784
pg_statistic  |  294912
(5 rows)
```

Если записать эту команду \set в файл ~/.psqlrc, переменная top5 будет доступна сразу после запуска psql.

Благодаря поддержке readline, в psql работает автодополнение ключевых слов и имен объектов, а также сохраняется история команд. Имя и размер файла истории настраиваются переменными HISTFILE, HISTSIZE.

psql — терминальный клиент для работы с СУБД

При запуске требуются параметры подключения

Выполняет команды SQL и psql

Содержит инструменты для интерактивной работы, а также для подготовки и выполнения скриптов

1. Запустите `psql` и проверьте информацию о текущем подключении.
2. Выведите список баз данных в подробном виде.
3. По умолчанию `psql` использует команду «`less`» для постраничного просмотра результатов запроса. Замените ее на команду «`less -XS`» и снова выведите подробный список баз данных.
4. По умолчанию приглашение `psql` показывает только имя базы данных. Настройте вид приглашения так: `пользователь@база=#`.
5. Настройте `psql` так, чтобы для всех команд выводилась длительность выполнения. Убедитесь, что при повторном запуске эта настройка сохраняется.

1. При запуске `psql` можно не указывать параметры подключения, будут действовать значения по умолчанию.

2. Используйте команду `\+`.

3. Программа постраничного просмотра настраивается переменной окружения `PSQL_PAGER`. Настройку можно сделать в файле `.psqlrc` с помощью команды `\setenv`. Это позволит задать значение «`less -XS`» только для работы в `psql`, а в остальных случаях будут действовать обычные настройки ОС. По умолчанию `less` переносит длинные строки при просмотре и очищает свой вывод при выходе. Параметр `-XS` отменяет это поведение.

4. Описание настройки приглашения в документации:

<https://postgrespro.ru/docs/postgresql/16/app-psql#APP-PSQL-PROMPTING>

5. Команду `psql` для вывода длительности выполнения запросов можно найти в документации или с помощью команды `\?`.

## 1. Запуск psql и просмотр информации о подключении

```
student$ psql
```

```
=> \conninfo
```

```
You are connected to database "student" as user "student" via socket in
"/var/run/postgresql" at port "5432".
```

---

## 2. Постраничный просмотр с помощью less

Если результат запроса не помещается в окне терминала, psql передает его программе less. В ней можно просматривать результаты запроса, используя клавиши навигации. Команда h выводит справку по less, команда q завершает просмотр.

По умолчанию less переносит длинные строки, из-за чего результаты бывает сложно читать. Также при выходе из less ранее выведенный текст стирается.

Например, команда \l+ выведет подробную информацию о базах данных, но просматривать ее неудобно.

```
=> \l+
```

```
List of databases
```

Name	Owner	Encoding	Locale Provider	Collate	Ctype	ICU
Locale	ICU Rules	Access privileges	Size	Tablespace		
Description						
postgres	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
connection database			7361 kB	pg_default	default administrative	
student	student	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
	=c/postgres		7361 kB	pg_default	unmodifiable empty database	
template1	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
	=c/postgres		7516 kB	pg_default	default template for new	
databases						
	postgres=CtC/postgres					

(4 rows)

---

## 3. Настройка постраничного просмотра в .psqlrc

Если использовать утилиту less с параметрами -XS, длинные строки не будут разбиваться переносами, а при выходе из less предыдущий вывод будет сохраняться. Для такой настройки достаточно установить переменную PSQL\_PAGER с помощью setenv. Запишем эту настройку в скрипт ~/.psqlrc:

```
student$ echo "\setenv PSQL_PAGER 'less -XS'" > ~/.psqlrc
```

## 4. Настройка приглашения

Для добавления информации о роли нужно в начало переменных PROMPT1 и PROMPT2 добавить %n@.

```
student$ echo "\set PROMPT1 '%n@%/%R%x%# '" >> ~/.psqlrc
```

```
student$ echo "\set PROMPT2 '%n@%/%R%x%# '" >> ~/.psqlrc
```

Переменная PROMPT1 определяет приглашение для первой строки вводимого пользователем запроса. Если запрос занимает более одной строки, начиная со второй за приглашение отвечает переменная PROMPT2. Обе переменные имеют одинаковое значение по умолчанию, но можно установить разные приглашения для первой и последующих строк запроса. Переменная PROMPT3 используется только для команды COPY.

## 5. Вывод длительности выполнения команд SQL

```
student$ echo '\timing on' >> ~/.psqlrc
```

В итоге содержимое файла .psqlrc станет таким:

```
student$ cat ~/.psqlrc
```

```
\setenv PSQL_PAGER 'less -XS'  
\set PROMPT1 '%n@%/%R%x%# '  
\set PROMPT2 '%n@%/%R%x%# '  
\timing on
```

Чтобы изменения вступили в силу, нужно выйти и заново войти в psql.

```
=> \q
```

```
student$ psql
```

Проверьте после повторного запуска:

- приглашение (должно включать имя роли);
- вывод подробной информации о базах данных;
- вывод времени выполнения команд.

1. Откройте транзакцию и выполните команду, которая завершается любой ошибкой. Убедитесь, что продолжить работу в этой транзакции невозможно.
2. Задайте переменной `ON_ERROR_ROLLBACK` значение `on` и убедитесь, что после ошибки можно выполнять команды внутри транзакции.

1. Чтобы открыть транзакцию, выполните команду

`BEGIN;`

2. Значение `ON_ERROR_ROLLBACK = on` заставляет `psql` устанавливать точку сохранения (`SAVEPOINT`) перед каждой командой SQL в открытой транзакции и в случае ошибки откатываться к этой точке сохранения.

<https://postgrespro.ru/docs/postgresql/16/sql-savepoint>

## 1. Утилита psql и обработка ошибок внутри транзакций

```
student$ psql
```

Утилита psql по умолчанию работает в режиме автоматической фиксации транзакций. Поэтому любая команда SQL выполняется в отдельной транзакции.

Чтобы явно начать транзакцию, нужно выполнить команду BEGIN:

```
student@student=# BEGIN;
```

```
BEGIN
```

Обратите внимание на то, что приглашение psql изменилось. Символ «звездочка» говорит о том, что транзакция сейчас активна.

```
student@student=# CREATE TABLE t (id int);
```

```
CREATE TABLE
```

Предположим, мы случайно сделали ошибку в следующей команде:

```
student@student=# INSERTINTO t VALUES(1);
```

```
ERROR: syntax error at or near "INSERTINTO"
LINE 1: INSERTINTO t VALUES(1);
      ^
```

О случившейся ошибке можно узнать из приглашения: звездочка изменилась на восклицательный знак. Попробуем исправить команду:

```
student@student=#!# INSERT INTO t VALUES(1);
```

```
ERROR: current transaction is aborted, commands ignored until end of transaction block
```

Но PostgreSQL не умеет откатывать только одну команду транзакции, поэтому транзакция обрывается и откатывается целиком. Чтобы продолжить работу, мы должны выполнить команду завершения транзакции. Не важно, будет ли это COMMIT или ROLLBACK, ведь транзакция уже отменена.

```
student@student=#!# COMMIT;
```

```
ROLLBACK
```

Создание таблицы было отменено, поэтому ее нет в базе данных:

```
student@student=# SELECT * FROM t;
```

```
ERROR: relation "t" does not exist
LINE 1: SELECT * FROM t;
      ^
```

## 2. Переменная ON\_ERROR\_ROLLBACK

Изменим поведение psql.

```
student@student=# \set ON_ERROR_ROLLBACK on
```

Теперь перед каждой командой транзакции неявно будет устанавливаться точка сохранения, а в случае ошибки будет происходить автоматический откат к этой точке. Это даст возможность продолжить выполнение команд транзакции.

```
student@student=# BEGIN;
```

```
BEGIN
```

```
student@student=# CREATE TABLE t (id int);
```

```
CREATE TABLE
```

```
student@student=# INSERTINTO t VALUES(1);
```

```
ERROR: syntax error at or near "INSERTINTO"
LINE 1: INSERTINTO t VALUES(1);
      ^
```

```
student@student=# INSERT INTO t VALUES(1);
```

```
INSERT 0 1
```

```
student@student=# COMMIT;
```

```
COMMIT
```



```
student@student=# SELECT * FROM t;
```

```
id  
----  
1  
(1 row)
```

Переменной ON\_ERROR\_ROLLBACK можно установить значение interactive, тогда подобное поведение будет только в интерактивном режиме работы, но не при выполнении скриптов.

```
student@student=# DROP TABLE t;
```

```
DROP TABLE
```