

# Базовый инструментарий Использование psql



## Авторские права

© Postgres Professional, 2015–2022

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

## 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 тем, кто привык работать с графическими инструментами; тем не менее, он весьма удобен, если к нему привыкнуть.

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/13/app-psql.html>

## 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 нужно указать параметры подключения.

К обязательным параметрам подключения относятся: 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* — локальное соединение;
- *порт* — обычно 5432.

Если требуется выполнить новое подключение не выходя из psql, нужно выполнить команду `\connect`.

Команда `\conninfo` выдает информацию о текущем подключении.

Дополнительная информация о возможностях настройки подключения:

<https://postgrespro.ru/docs/postgresql/13/libpq-envvars.html>

<https://postgrespro.ru/docs/postgresql/13/libpq-pgservice.html>

<https://postgrespro.ru/docs/postgresql/13/libpq-pgpass.html>

## В командной строке ОС

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

## В psql

=> \?	список команд psql
=> \? variables	переменные psql
=> \h[elp]	список команд SQL
=> \h <i>команда</i>	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.

Внутри psql есть возможность получить список и краткое описание команд psql. Все команды psql начинаются с обратной косой черты.

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

И еще одна команда, которую полезно знать, хотя она и не имеет отношения к справке: это `\q` — выход из psql. В качестве альтернативы для выхода можно также 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 \c[onnect] выполняет новое подключение, не покидая psql.

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

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

Команды SQL, в отличие от команд psql, могут располагаться на нескольких строках. Для отправки команды SQL на выполнение завершаем команду 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)

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 режим 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
```

## Взаимодействие с ОС

В psql можно выполнять команды shell:

```
=> \! pwd
```

```
/home/student
```

Можно установить переменную окружения операционной системы:

```
=> \setenv TEST Hello
```

```
=> \! echo $TEST
```

```
Hello
```

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

```
=> \o dbal_log
```

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

На экран ничего не попало. Посмотрим в файле:

```
=> \! cat dbal_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. В конце указано имя файла, в который будет записан вывод.

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

Вот что получилось в файле:

```
=> \! cat dbal_log
```

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

Вывод запроса можно перенаправить команде ОС, если указать `\g | cmd`

Выполняем теперь эти команды с помощью `\i[nclude]`:

```
=> \i dba1_log

count
-----
  402
(1 row)

count
-----
  411
(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
-----
  402
(1 row)

count
-----
  411
(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 определенный смысл для `psql`).

Установим переменную:

```
=> \set TEST Hi!
```

Чтобы получить значение, надо предварить имя переменной двоеточием:

```
=> \echo :TEST
```

Hi!

Значение переменной можно сбросить:

```
=> \unset TEST
```

```
=> \echo :TEST
```

:TEST

---

Можно результат запроса записать в переменную. Для этого запрос нужно завершить командой `\gset`:

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

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

2022-11-30 18:08:22.519029+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'
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 = '13.7 (Ubuntu 13.7-1.pgdg22.04+1)'
SERVER_VERSION_NUM = '130007'
SHOW_CONTEXT = 'errors'
SINGLELINE = 'off'
SINGLESTEP = 'off'
SQLSTATE = '00000'
USER = 'student'
VERBOSE = 'default'
VERSION = 'PostgreSQL 13.7 (Ubuntu 13.7-1.pgdg22.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0, 64-bit'
VERSION_NAME = '13.7 (Ubuntu 13.7-1.pgdg22.04+1)'
VERSION_NUM = '130007'
curr_time = '2022-11-30 18:08:22.519029+03'

```

В командных файлах 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 ОС
\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
```

Column	View "pg_catalog.pg_tables"
	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:

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

Например, 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_depend	1130496
pg_proc	1048576
pg_rewrite	704512
pg_attribute	671744
pg_description	581632

(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 все строки таблицы `pg_tables`.
3. Установите команду «`less -XS`» для постраничного просмотра и еще раз выведите все строки `pg_tables`.
4. Приглашение по умолчанию показывает имя базы данных. Настройте приглашение так, чтобы дополнительно выводилась информация о пользователе: `роль@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.

3. Настройку переменной окружения `PSQL_PAGER` можно сделать в файле `.psqlrc`. Используйте команду `\setenv`, чтобы установить переменную окружения ОС. Это позволит использовать значение «`less -XS`» только при работе в `psql`. Для всех остальных команд ОС будут использоваться настройки в ОС (например, из `.profile`).

4. Описание настройки приглашения в документации:

<https://postgrespro.ru/docs/postgresql/13/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. Таблица `pg_tables`

Ограничим выборку пятью записями. Обратите внимание: если записи не помещаются по ширине на экран, они переносятся на новые строки. Просматривать результаты запроса неудобно.

```
=> SELECT * FROM pg_tables LIMIT 5;
```

schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	hastriggers	rowsecurity
pg_catalog	pg_statistic	postgres		t	f	f	f
pg_catalog	pg_type	postgres		t	f	f	f
pg_catalog	pg_foreign_table	postgres		t	f	f	f
pg_catalog	pg_authid	postgres	pg_global	t	f	f	f
pg_catalog	pg_statistic_ext_data	postgres		t	f	f	f

(5 rows)

## 3. Настройка страничного просмотра в `.psqlrc`

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

Для просмотра результатов запроса в команде `less` можно использовать стрелки для прокрутки вниз, вверх, влево и вправо. Команда `h` выводит справку по `less`. Команда `q` завершает просмотр.

## 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);
- отображение результатов запроса из `pg_tables`;
- вывод времени выполнения команд.

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

1. Для открытия транзакции выполните команду

`BEGIN;`

2. Установка переменной `ON_ERROR_ROLLBACK` 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/13/sql-savepoint>

## 1. Утилита psql и обработка ошибок внутри транзакциях

```
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.