PostgreSQL 9.4

psql

Name

psql -- Интерактивный терминал PostgreSQL

Synopsis

`psql` [*option*...] [*dbname* [*username*]]

Описание

psql - это терминальный клиент для работы с PostgreSQL. Позволяет интерактивно вводить запросы, отправлять их в PostgreSQL и смотреть результаты. Ввод может быть не только интерактивным, но и из файла. Кроме того, предоставляется ряд мета-команд и различные возможности подобные тем, что имеются у командных оболочек, для облегчения написания скриптов и автоматизации широкого спектра задач.

Опции

`-a`
`--echo-all`

Отправляет на стандартный вывод все непустые входные строки по мере их чтения. (Это не относится к строкам, считанным в интерактивном режиме.) Эквивалентно установке переменной `ECHO` в значение `all`.

`-A`
`--no-align`

Переключение на невыровненный режим вывода. (По умолчанию, наоборот, используется выровненный режим вывода.)

`-c` *`command`*
`--command=`*`command`*

Указывает, что psql должен выполнить одну командную строку, *`command`*, а затем завершить работу. Это полезно в скриптах. Файлы запуска (`psqlrc` и `~/.psqlrc`) с этой опцией игнорируются.

`command` должна быть либо командной строкой, которая полностью интерпретируется сервером (т.е. не содержит специфических для psql возможностей), либо одиночной командой, начинающейся с \. Таким образом, при использовании этой опции нельзя

смешивать SQL и мета-команды psql. Тем не менее, этого можно добиться, если передать строку в psql вот так: `echo '\x \\ SELECT * FROM foo;' | psql`. (`\\` является разделителем для мета-команд.)

Если командная строка содержит несколько команд SQL, они обрабатываются в одной транзакции, если только нет явных `BEGIN`/`COMMIT` команд, включенных в строку, разделяющих ее на несколько транзакций. Это отличается от поведения, если эту же строку подать на стандартный ввод psql. Также, возвращается результат только последней SQL-команды.

Из-за такого унаследованного поведения, использование более одной команды в опции `-c` часто приводит к неожиданным результатам. Лучше подавать несколько команд на стандартный ввод psql, либо с использованием echo, как показано выше, либо через возможности командной оболочки ОС, например:

```
psql <<EOF
\x
SELECT * FROM foo;
EOF
```

`-d` *`dbname`*
`--dbname=`*`dbname`*

Указывает имя базы данных для подключения. Эквивалентно указанию *`dbname`* в качестве первого аргумента, не являющегося параметром в командной строке.

Если этот параметр содержит знак `=` или начинается с допустимого URI префикса (`postgresql://` или `postgres://`), он рассматривается как строка `conninfo`. Дополнительная информация в Section 31.1.1.

`-e`
`--echo-queries`

Посылает все команды SQL, отправленные на сервер, еще и на стандартный вывод. Эквивалентно установке переменной `ECHO` в значение `queries`.

`-E`
`--echo-hidden`

Отображает фактические запросы, генерируемые `\d` и другими командами, начинающимися с `\`. Это можно использовать для изучения внутренних операций в psql. Эквивалентно установке переменной `ECHO_HIDDEN` значения `on`.

`-f` *`filename`*
`--file=`*`filename`*

Использует файл *`filename`* в качестве источника команд вместо чтения команд в интерактивном режиме. После обработки файла, работа psql завершается. Это во многом эквивалентно мета-команде `\i`.

Если *filename* это `-` (дефис), то читается стандартный ввод.

Использование этой опции немного отличается от `psql < filename`. В основном, оба варианта будут делать то, что вы ожидаете, но с `-f` доступны некоторые полезные свойства, такие как сообщения об ошибках с номерами строк. Также есть небольшая вероятность, что с этой опцией уменьшаются накладные расходы на запуск. С другой стороны, вариант с перенаправлением ввода из командного интерпретатора (в теории) гарантирует получение точно такого же вывода, какой вы получили бы, если ввели все вручную.

`-F` *separator*
`--field-separator=`*separator*

Использование *separator* в качестве разделителя полей при невыровненном режиме вывода. Эквивалентно `\pset fieldsep` или `\f`.

`-h` *hostname*
`--host=`*hostname*

Указывает имя хоста машины, на которой запущен сервер. Если значение начинается с косой черты, то оно используется в качестве директории для доменного сокета Unix.

`-H`
`--html`

Включает табличный вывод в формате HTML. Эквивалентно `\pset format html` или команде `\H`.

`-l`
`--list`

Выводит список всех доступных баз данных и завершает работу. Другие опции, не связанные с соединением, игнорируются. Это похоже на мета-команду `\list`.

`-L` *filename*
`--log-file=`*filename*

В дополнение к обычному выводу, записывает вывод результатов всех запросов в файл *filename*.

`-n`
`--no-readline`

Отключает использование Readline для редактирования командной строки и использования истории команд. Может быть полезно, для выключения расширенных действий клавиши табуляции при вырезании и вставке.

`-o` *filename*
`--output=`*filename*

Записывает вывод результатов всех запросов в файл *filename*. Эквивалентно команде `\o`.

`-p` *port*
`--port=`*port*

Задает порт TCP или локальный доменный сокет Unix, на котором сервер прослушивает соединения. По умолчанию используется значение переменной среды `PGPORT`, если оно не установлено, то используется значение, указанное во время компиляции, обычно 5432.

`-P` *assignment*
`--pset=`*assignment*

Задает параметры печати, в стиле команды `\pset`. Обратите внимание, что имя параметра и значение разделяются знаком равенства, а не пробела. Например, чтобы установить формат вывода в LaTeX, нужно написать `-P format=latex`.

`-q`
`--quiet`

Указывает, что psql должен работать без вывода дополнительных сообщений. По умолчанию, выводятся приветствия и различные информационные сообщения. Этого не произойдет с использованием данной опции. Полезно вместе с опцией `-c`. Это эквивалентно установке переменной `QUIET` значения `on`.

`-R` *separator*
`--record-separator=`*separator*

Использует *separator* как разделитель записей при невыровненном режиме вывода. Эквивалентно команде `\pset recordsep`.

`-s`
`--single-step`

Запуск в пошаговом режиме. Это означает, что пользователь будет подтверждать выполнение каждой команды, отправляемой на сервер, с возможностью отменить выполнение. Используется для отладки скриптов.

`-S`
`--single-line`

Запуск в однострочном режиме, при котором символ новой строки завершает SQL команды, также как это делает точка с запятой.

> **Note:** Этот режим предназначен для тех, кто на нем настаивает, но вы не обязаны рекомендовать его использовать. В частности, если смешать на одной строке команды SQL и мета-команды, то порядок их выполнения может быть не всегда очевиден для неопытного пользователя.

`-t`
`--tuples-only`

Отключает вывод имен столбцов и результирующей строки с количеством выбранных записей. Эквивалентно команде `\t`.

`-T` *`table_options`*
`--table-attr=`*`table_options`*

Задает атрибуты, которые будут вставлены в тег HTML `table`. Смотри `\pset` для деталей.

`-U` *`username`*
`--username=`*`username`*

Подключение к базе данных под пользователем *`username`* вместо используемого по умолчанию. (Разумеется, при наличии прав на это.)

`-v` *`assignment`*
`--set=`*`assignment`*
`--variable=`*`assignment`*

Выполняет присвоение значения переменной, как мета-команда `\set`. Обратите внимание на то, что необходимо разделить имя переменной и значение (при наличии) знаком равенства в командной строке. Чтобы сбросить переменную, опустите знак равенства. Чтобы установить пустое значение, поставьте знак равенства, но опустите значение. Присваивания выполняются на очень ранней стадии запуска, поэтому если переменные зарезервированы для внутренних целей, то позже они могут быть перезаписаны.

`-V`
`--version`

Выводит версию psql и завершает работу.

`-w`
`--no-password`

Никогда не выдавать запрос на ввод пароля. Если сервер требует аутентификацию по паролю и пароль не доступен с помощью других средств, таких как файл `.pgpass`, попытка соединения не удастся. Опция может быть полезна в пакетных заданиях и скриптах, где нет пользователя, который вводит пароль.

Обратите внимание, что эта опция действует на протяжении всей сессии и, таким образом, влияет на мета-команду `\connect`, также как и на первую попытку соединения.

`-W`
`--password`

Принудительно запрашивает пароль перед подключением к базе данных.

Эта опция не существенна, так как psql автоматически запрашивает пароль, если сервер требует аутентификацию по паролю. Однако, psql использует дополнительную попытку соединения для выяснения того, что серверу требуется пароль. В некоторых случаях стоит вводить `-W`, чтобы избежать лишней попытки соединения.

Обратите внимание, что эта опция действует на протяжении всей сессии и, таким образом, влияет на мета-команду `\connect`, также как и на первую попытку соединения.

`-x`
`--expanded`

Включает режим развернутого вывода таблицы. Эквивалентно команде `\x`.

`-X,`
`--no-psqlrc`

Не читать стартовые файлы (ни общесистемный файл `psqlrc`, ни пользовательский файл `~/.psqlrc`).

`-z`
`--field-separator-zero`

Установить нулевой байт в качестве разделителя полей для невыровненного режима вывода.

`-0`
`--record-separator-zero`

Установить нулевой байт в качестве разделителя записей для невыровненного режима вывода. Это полезно при взаимодействии с другими программами, например, с `xargs -0`.

`-1`
`--single-transaction`

Если psql выполняет скрипт, то добавление этой опции заключает скрипт в `BEGIN`/`COMMIT` для выполнения в рамках одной транзакции. Это гарантирует, что либо все команды успешно завершены, либо никаких изменений не произведено.

Если в самом скрипте используются `BEGIN`, `COMMIT` или `ROLLBACK`, то эта опция не будет иметь желаемого эффекта. Кроме того, если скрипт содержит любую команду, которая не может быть выполнена внутри транзакционного блока, указание этой опции приведет к сбою команды и, следовательно, всей транзакции.

`-?`
`--help`

Показывает справку об аргументах командной строки psql и завершает работу.

Код завершения

При нормальном завершении psql возвращает 0 в командную оболочку ОС, 1 - если произошла фатальная ошибка в самом psql (например, нехватка памяти, файл не найден), 2 - при неудачном соединении с сервером неинтерактивного сеанса, 3 - при ошибке в скрипте и установленной переменной `ON_ERROR_STOP`.

Использование

Подключение к базе данных

psql это клиент для PostgreSQL. Для подключения к базе данных нужно знать имя базы данных, имя хоста, номер порта сервера и имя пользователя, под которым вы хотите подключиться. Эти параметры можно задать через опции командной строки, а именно `-d`, `-h`, `-p` и `-U` соответственно. Если в командной строке есть аргумент, который не принадлежит ни к одной опции psql, то он используется в качестве имени базы данных (или имени пользователя, если база данных уже задана). Не все эти опции необходимы, есть полезные значения по умолчанию. Если опустить имя хоста, psql будет подключаться через сокет Unix-домена к серверу на локальном хосте, или подключаться к `localhost` через TCP/IP на компьютерах, не использующих UNIX сокеты. Номер порта по умолчанию определяется во время компиляции. Поскольку сервер базы данных использует то же значение по умолчанию, в большинстве случаев вам не нужно указывать номер порта. Имя пользователя по умолчанию, как и имя базы данных по умолчанию, совпадает с именем пользователя в операционной системе. Обратите внимание, что вы не можете просто подключиться к любой базе данных под любым именем пользователя. Администратор базы данных должен сообщить вам о ваших правах доступа.

Если значения по умолчанию не подходят, можно сэкономить на вводе параметров подключения установив переменные среды `PGDATABASE`, `PGHOST`, `PGPORT` и/или `PGUSER`. (О дополнительных переменных среды, смотри Section 31.14.) Также удобно иметь файл `~/.pgpass` для избегания постоянного ввода паролей. Дополнительная информация в Section 31.15.

Альтернативный способ указать параметры подключения это использование строки `conninfo` или URI вместо имени базы данных. Этот механизм дает широкие возможности по управлению параметрами подключения. Например:

```
$ psql "service=myservice sslmode=require"
$ psql postgresql://dbmaster:5433/mydb?sslmode=require
```

Этот способ также позволяет использовать LDAP для получения параметров подключения, как описано в Section 31.17. Смотри Section 31.1.2 для информации обо всех имеющихся опциях соединения.

Если соединение не может быть установлено по любой причине (например, нет прав, сервер не работает и т.д.), psql вернет ошибку и прекратит работу.

Если и стандартный ввод, и стандартный вывод являются терминалом, то psql установит кодировку клиента в "auto", и подходящая клиентская кодировка будет определяться из локальных установок (переменная окружения `LC_CTYPE` на Unix системах). Если это работает не так как ожидалось, кодировку клиента можно изменить, установив переменную окружения `PGCLIENTENCODING`.

Ввод SQL команд

Как правило, приглашение psql состоит из имени базы данных, к которой psql в данный момент подключен, а затем строки `=>`. Например:

```
$ psql testdb
psql (9.4.1)
Введите "help", чтобы получить справку.

testdb=>
```

В командной строке, пользователь может вводить команды SQL. Обычно, введенные строки отправляются на сервер когда встречается точка с запятой, завершающая команду. Конец строки не завершает команду. Это позволяет разбивать команды на несколько строк для лучшего понимания. Если команда была отправлена и выполнена без ошибок, то результат команды выводится на экран.

При каждом выполнении команды, psql также проверяет асинхронные уведомления о событиях, генерируемые командами LISTEN и NOTIFY.

Комментарии в стиле C передаются для обработки на сервер, в то время как комментарии в стандарте SQL psql удаляет перед отправкой.

Мета-Команды

Всё что вводится в psql не взятое в кавычки и начинающееся с обратной косой черты, является мета-командой psql и обрабатывается самим psql. Эти команды делают psql полезным для задач администрирования и разработки скриптов.

Формат команды psql следующий: бэкслеш, сразу за ним команда, затем аргументы. Аргументы отделяются от команды и друг от друга любым количеством пробелов.

Чтобы включить пробел в значение аргумента нужно заключить его в одинарные кавычки. Чтобы включить одинарную кавычку в значение аргумента нужно написать две одинарные кавычки внутри текста в одинарных кавычках. Всё, что содержится в одинарных кавычках подлежит заменам принятым в языке C: `\n` (новая строка), `\t` (табуляция), `\b` (backspace), `\r` (возврат каретки), `\f` (подача страницы), `\`*цифры* (восьмеричное число), и `\x`*цифры* (шестнадцатеричное число). Если внутри текста в одинарных кавычках встречается бэкслеш перед любым другим символом, то он экранирует этот символ.

Текст аргумента, заключенный в обратные кавычки (`) считается командной строкой, которая передается в командную оболочку ОС. Вывод от этой команды (с удаленными в конце символами новой строки) заменяет текст в обратных кавычках.

Если внутри аргумента встречается не взятое в кавычки имя psql переменной с двоеточием (:) перед ним, то оно заменяется на значение psql переменной, как описано в *Интерполяция SQL*.

Некоторые команды принимают идентификатор SQL (например, имя таблицы) в качестве аргумента. Такие аргументы следуют правилам синтаксиса SQL: буквы, не взятые в кавычки, преобразуются в нижний регистр, буквы, взятые в двойные кавычки (") предотвращают преобразование регистра и позволяют включать пробелы в идентификатор. Внутри двойных кавычек, две двойные кавычки сокращаются до одной. Например, `FOO"BAR"BAZ` интерпретируется как `fooBARbaz`, а `"A weird"" name"` становится `A weird" name`.

Разбор аргументов останавливается в конце строки или когда встречается другой, не внутри кавычек, бэкслеш. Бэкслеш не внутри кавычек, рассматривается как начало новой мета-команды. Специальная последовательность `\\` (два бэкслеша) обозначает окончание аргументов, далее продолжается разбор команд SQL, если таковые имеются. Таким образом, команды SQL и psql можно свободно смешивать на одной строке. Но в любом случае, аргументы мета-команды не могут выходить за пределы текущей строки.

Определены следующие мета-команды:

`\a`

> Если текущий режим вывода таблицы невыровненный, то он переключается на выравненный режим. Если текущий режим выровненный, то устанавливается невыровненный. Эта команда поддерживается для обратной совместимости. См. `\pset` для более общего решения.

`\c` или `\connect [ dbname [ username ] [ host ] [ port ] ]`

> Устанавливает новое подключение к серверу PostgreSQL. Если новое подключение успешно установлено, предыдущее подключение закрывается. Если какой-либо из параметров подключения *dbname*, *username*, *host* или *port* не указан или указан как `-`, используется значение этого параметра от предыдущего соединения. Если нет предыдущего подключения, используются значения по умолчанию принятые в libpq.
>
> Если попытка подключения не удалась (неверное имя пользователя, доступ запрещен, и т.д.), то предыдущее соединение останется активным, если psql находится в интерактивном режиме. Если скрипт выполняется не интерактивно, обработка немедленно останавливается с сообщением об ошибке. Такое различие в поведении было выбрано для удобства пользователя в отношении опечаток с одной стороны, и механизма безопасности, при котором скрипты не будут запущены на неправильной базе данных с другой стороны.

`\C [ title ]`

Задаёт заголовок, который будет выводиться для результатов любых запросов или отменяет установленный ранее заголовок. Эта команда эквивалентна `\pset title` *`title`*. (Название этой команды происходит от "caption", т.к. ранее это использовалось только для задания заголовков HTML таблиц.)

`\cd [` *`directory`* `]`

Заменяет текущий рабочий каталог на *`directory`*. Без аргумента, устанавливается домашний каталог текущего пользователя.

> **Tip:** для печати текущего рабочего каталога, используйте `\! pwd`.

`\conninfo`

Выводит информацию о текущем подключении к базе данных.

`\copy { `*`table`*` [ ( `*`column_list`*` ) ] | ( `*`query`*` ) } { from | to } { `*`'filename'`*` | program `*`'command'`*` | stdin | stdout | pstdin | pstdout } [ [ with ] ( `*`option`*` [, ...] ) ]`

Выполняет копирование на клиенте. Это операция, которая выполняет SQL команду COPY, но вместо чтения или записи в файл на сервере, psql читает или записывает файл и пересылает данные между сервером и локальной файловой системой. Это означает, что для доступа к файлам используются привилегии локального пользователя, а не сервера, и не требуются привилегии суперпользователя SQL.

Когда указана `program`, psql выполняет *`command`* и данные из/в *`command`* передаются между сервером и клиентом. Это опять же означает, что для выполнения программ используются привилегии локального пользователя, а не сервера, и не требуются привилегии суперпользователя SQL.

При выполнении `\copy ... from stdin` строки с данными считываются из источника, выполневшего команду, и считываются до тех пор, пока не встретится `\.` или не будет достигнут конец файла. Эта опция полезна для заполнения таблиц прямо в SQL скриптах. При выполнении `\copy ... to stdout` вывод направляется в тоже место, что и вывод psql команд. Статус команды `COPY` *`count`* не отображается, чтобы не перепутать со строкой данных. Для чтения/записи стандартного ввода/вывода psql, вне зависимости от источника текущей команды или опции `\o`, используйте `from pstdin` или `to pstdout`.

Синтаксис команды похож на синтаксис SQL команды COPY. Все опции, кроме источника и получателя данных, соответствуют опциям COPY. Поэтому к команде `\copy` применяются специальные правила разбора. В частности, не применяются правила подстановки переменных и экранирование при помощи бэкслеш.

> **Tip:** Эта операция не так эффективна, как SQL команда `COPY`, потому что все данные перемещаются между клиентом и сервером. Для больших объемов данных SQL команда может быть предпочтительнее.

`\copyright`

Показывает информацию об авторских правах и условиях распространения PostgreSQL.

`\d[S+] [ pattern ]`

Для каждого отношения (таблицы, представления, индекса, последовательности, внешней таблицы) или составного типа, соответствующих шаблону *`pattern`*, показывает столбцы, их типы, табличное пространство (если не по умолчанию) и любые специальные атрибуты, такие как `NOT NULL` или значения по умолчанию. Также показываются связанные индексы, ограничения, правила и триггеры. Для внешних таблиц показывается связанный внешний сервер. ("Соответствие шаблону" определяется ниже в *Шаблоны поиска (Patterns)*.)

Для некоторых типов отношений, `\d` показывает дополнительную информацию по каждому столбцу: значения столбца для последовательностей, индексное выражение для индексов и параметры обработчика внешних данных для внешних таблиц.

Вариант команды `\d+` похож на `\d`, но выводит больше информации: комментарии к столбцам таблицы, наличие в таблице OID, для представлений показывается его определение, отличные от значений по умолчанию установки replica identity.

По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор `S`.

> **Note:** Если `\d` используется без *`pattern`*, это эквивалентно `\dtvsE` и показывает список всех доступных таблиц, представлений, последовательностей и внешних таблиц. Чисто для удобства.

`\da[S] [ pattern ]`

Выводит список агрегатных функций, вместе с типом возвращаемого значения и типами данных, которыми они оперируют. Если указан *`pattern`*, отображаются только функции, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор `S`.

`\db[+] [ pattern ]`

Выводит список табличных пространств. Если указан *`pattern`*, отображаются только табличные пространства, имена которых соответствуют шаблону. При добавлении `+` к имени команды, для каждого объекта дополнительно будут выводиться права доступа и описание.

`\dc[S+] [ pattern ]`

Выводит список преобразований между кодировками наборов символов. Если указан *`pattern`*, отображаются только преобразования кодировок, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные

пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого объекта дополнительно будет выводиться описание.

`\dC[+] [ pattern ]`

Выводит список приведения типов. Если указан *pattern*, отображаются только приведения типов, имена которых соответствуют шаблону. При добавлении + к имени команды, для каждого объекта дополнительно будет выводиться описание.

`\dd[S] [ pattern ]`

Показывает описания объектов следующих типов: `constraint`, `operator class`, `operator family`, `rule` и `trigger`. Описания для остальных объектов можно посмотреть соответствующими мета-командами для этих типов объектов.

`\dd` показывает описания для объектов, соответствующих шаблону *pattern*, или для доступных объектов указанных типов, если аргументы не заданы. Но в любом случае, выводятся только те объекты, которые имеют описание. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S.

Описания объектов создаются SQL командой COMMENT.

`\ddp [ pattern ]`

Выводит список настроек прав доступа по умолчанию. Выводится строка для каждой роли (и схемы, если применимо), для которой настройки прав доступа по умолчанию были изменены от встроенных по умолчанию. Если указан *pattern*, выводятся строки только для тех ролей и схем, чьи имена соответствуют шаблону.

Права доступа по умолчанию устанавливаются командой ALTER DEFAULT PRIVILEGES. Смысл отображаемых привилегий объясняется в GRANT.

`\dD[S+] [ pattern ]`

Выводит список доменов. Если указан *pattern*, отображаются только домены, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого объекта дополнительно будут выводиться права доступа и описание.

```
\dE[S+] [ pattern ]
\di[S+] [ pattern ]
\dm[S+] [ pattern ]
\ds[S+] [ pattern ]
\dt[S+] [ pattern ]
\dv[S+] [ pattern ]
```

В этой группе команд буквы `E`, `i`, `m`, `s`, `t` и `v` обозначают соответственно: внешнюю таблицу, индекс, материализованное представление, последовательность, таблицу и представление. Можно указывать все или часть этих букв, в произвольном порядке, чтобы получить список объектов этих типов. Например, `\dit` выводит список индексов и таблиц. При добавлении `+` к имени команды, для каждого объекта дополнительно будут выводиться физический размер на диске и описание, при наличии. Если указан *`pattern`*, отображаются только объекты, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор `S`.

`\des[+] [ pattern ]`

Выводит список внешних серверов (мнемоника: "external servers"). Если указан *`pattern`*, отображаются только сервера, имена которых соответствуют шаблону. Если используется форма `\des+`, то выводится полное описание каждого сервера, включая права доступа, тип, версию, параметры и описание.

`\det[+] [ pattern ]`

Выводит список внешних таблиц (мнемоника: "external tables"). Если указан *`pattern`*, выводятся только те записи, у которых имя таблицы или схемы соответствуют шаблону. Если используется форма `\det+`, то дополнительно выводятся общие параметры и описание внешней таблицы.

`\deu[+] [ pattern ]`

Выводит список сопоставлений пользователей (мнемоника: "external users"). Если указан *`pattern`*, отображаются только сопоставления, у которых имена пользователей соответствуют шаблону. Если используется форма `\deu+`, то выводится дополнительная информация о каждом сопоставлении пользователей.

Caution

`\deu+` также может отображать имя и пароль удаленного пользователя, поэтому следует позаботиться о том, чтобы не раскрывать их.

`\dew[+] [ pattern ]`

Выводит список обработчиков внешних данных (мнемоника: "external wrappers"). Если указан *`pattern`*, отображаются только обработчики внешних данных, имена которых соответствуют шаблону. Если используется форма `\dew+`, то дополнительно выводятся права доступа, параметры и описание обработчика.

`\df[antwS+] [ pattern ]`

Возвращает список функций, вместе с их аргументами, возвращаемыми типами и типами функций, которые классифицируются как "agg" (агрегатная), "normal" (обычная), "trigger" (триггерная) или "window" (оконная). Чтобы отобразить только функции определенного типа(типов), добавьте в команду соответствующие буквы `a`, `n`,

`t` или `w`. Если указан pattern, отображаются только функции, имена которых соответствуют шаблону. По умолчанию выводятся только функции созданные пользователем, для включения системных объектов нужно задать шаблон или добавить модификатор `S`. Если используется форма `\df+`, то выводится дополнительная информация о каждой функции: безопасность, волатильность, владелец, язык, исходный код и описание.

> **Tip:** Чтобы найти функции с аргументами или возвращаемыми значениями определенного типа, используйте возможности поиска вашего пейджера для прокрутки вывода команды `\df`.

`\dF[+] [ pattern ]`

Выводит список конфигураций текстового поиска. Если указан *`pattern`*, отображаются только конфигурации, имена которых соответствуют шаблону. Если используется форма `\dF+`, то выводится полное описание для каждой конфигурации, включая базовый синтаксический анализатор и используемые словари для каждого типа токена.

`\dFd[+] [ pattern ]`

Выводит список словарей текстового поиска. Если указан *`pattern`*, отображаются только словари, имена которых соответствуют шаблону. Если используется форма `\dFd+`, то выводится дополнительная информация о каждом словаре включая базовый шаблон текстового поиска и параметры инициализации.

`\dFp[+] [ pattern ]`

Выводит список анализаторов текстового поиска. Если указан *`pattern`*, отображаются только анализаторы, имена которых соответствуют шаблону. Если используется форма `\dFp+`, то выводится полное описание для каждого анализатора, включая базовые функции и список типов токенов.

`\dFt[+] [ pattern ]`

Выводит список шаблонов текстового поиска. Если указан *`pattern`*, отображаются только шаблоны, имена которых соответствуют шаблону. Если используется форма `\dFt+`, то выводится дополнительная информация о каждом шаблоне, включая имена основных функций.

`\dg[+] [ pattern ]`

Выводит список ролей базы данных. (Так как понятия "пользователи" и "группы" были объединены в "роли", эта команда теперь эквивалентна `\du`.) Если указан *`pattern`*, отображаются только роли, имена которых соответствуют шаблону. Если используется форма `\dg+`, то выводится дополнительная информация о каждой роли, в настоящее время добавлено описание роли.

`\dl`

Это псевдоним для `\lo_list`, показывает список больших объектов.

`\dL[S+] [ pattern ]`

Выводит список процедурных языков. Если указан *`pattern`*, отображаются только языки, имена которых соответствуют шаблону. По умолчанию отображаются только языки, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого языка дополнительно будут выводиться: обработчик вызова, валидатор, права доступа и является ли язык системным объектом.

`\dn[S+] [ pattern ]`

Выводит список схем (пространств имён). Если указан *`pattern`*, отображаются только схемы, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого объекта дополнительно будут выводиться права доступа и описание, при наличии.

`\do[S+] [ pattern ]`

Выводит список операторов, их операндов и типы результата. Если указан *`pattern`*, отображаются только операторы, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого оператора будет выводиться дополнительная информация, сейчас это имя функции, на которой основан оператор.

`\dO[S+] [ pattern ]`

Выводит список правил сортировки. Если указан *`pattern`*, отображаются только правила, имена которых соответствуют шаблону. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S. При добавлении + к имени команды, для каждого объекта дополнительно будет выводиться описание, при наличии. Обратите внимание, что отображаются только правила сортировки, применимые к кодировке текущей базы данных, поэтому результат команды может отличаться для различных баз данных этой же установки PostgreSQL.

`\dp [ pattern ]`

Выводит список таблиц, представлений и последовательностей с их правами доступа. Если указан *`pattern`*, отображаются только таблицы, представления и последовательности, имена которых соответствуют шаблону.

Для установки прав доступа используются команды GRANT и REVOKE. Смысл отображаемых привилегий объясняется в GRANT.

`\drds [ role-pattern [ database-pattern ] ]`

Выводит список специфических параметров конфигурации. Эти параметры могут быть специфическими для роли, специфическими для базы данных, или обеих. *role-pattern* и *database-pattern* используются для отбора по конкретным ролям и базам данных. Если они опущены, или указано *, выводятся все параметры конфигурации, в том числе не относящиеся к ролям или базам данных.

Команды ALTER ROLE и ALTER DATABASE используются для определения параметров конфигурации специфических для роли или базы данных.

`\dT[S+] [ pattern ]`

Выводит список типов данных. Если указан *pattern*, отображаются только типы, имена которых соответствуют шаблону. При добавлении + к имени команды, для каждого типа данных дополнительно будет выводиться: внутреннее имя типа, размер, допустимые значения для типа enum и права доступа. По умолчанию отображаются только объекты, созданные пользователем. Для включения системных объектов нужно задать шаблон или добавить модификатор S.

`\du[+] [ pattern ]`

Выводит список ролей базы данных. (Так как понятия "пользователи" и "группы" были объединены в "роли", эта команда теперь эквивалентна \dg.) Если указан *pattern*, отображаются только роли, имена которых соответствуют шаблону. Если используется форма \du+, то выводится дополнительная информация о каждой роли, в настоящее время добавлено описание роли.

`\dx[+] [ pattern ]`

Выводит список установленных расширений. Если указан *pattern*, отображаются только расширения, имена которых соответствуют шаблону. Если используется форма \dx+, то для каждого расширения выводятся все принадлежащие ему объекты.

`\dy[+] [ pattern ]`

Выводит список триггеров событий. Если указан *pattern*, отображаются только триггеры событий, имена которых соответствуют шаблону. При добавлении + к имени команды, для каждого объекта дополнительно будет выводиться описание.

`\e` или `\edit [ filename ] [ line_number ]`

Если указано имя файла *filename*, файл открывается для редактирования. После выхода из редактора, его содержимое копируется в буфер запроса. Если не указано имя файла, текущий буфер запроса копируется во временный файл и открывается в редакторе.

Затем новый буфер запроса повторно анализируется согласно обычным правилам psql, при этом весь буфер рассматривается как одна строка. (Таким образом, это не подходит для создания скриптов. Для скриптов используйте \i.) Если запрос заканчивается (или содержит) точкой с запятой, он немедленно выполняется. В противном случае он

просто будет ждать в буфере запроса. Введите точку с запятой или `\g` для отправки на выполнение или `\r` для отмены.

Если указан номер строки, psql будет позиционировать курсор на указанную строку файла или буфера запроса. Обратите внимание, что если указан один аргумент и он числовой, psql предполагает, что это номер строки, а не имя файла.

> **Tip:** См. ниже в разделе *Переменные окружения* о том, как настроить редактор.

`\echo` *`text`* `[ ... ]`

Выводит аргументы на стандартный вывод, разделяя их одним пробелом, в конце следует перевод строки. Команда полезна для формирования вывода из скриптов. Например:

```
=> \echo `date`
Tue Oct 26 21:40:57 CEST 1999
```

Если первый аргумент `-n` без кавычек, то перевод строки в конце не ставится.

> **Tip:** Если используется команда `\o` для перенаправления вывода запросов, возможно, следует применять команду `\qecho` вместо этой.

`\ef [` *`function_description`* `[` *`line_number`* `] ]`

Эта команда извлекает из базы данных определение заданной функции, в форме `CREATE OR REPLACE FUNCTION`, и отправляет его на редактирование. Редактирование осуществляется таким же образом, как и для `\edit`. После выхода из редактора, измененная команда будет находиться в буфере запроса. Введите точку с запятой или `\g` для выполнения или `\r` для отмены.

Для функции может быть задано только имя или имя и аргументы, например `foo(integer, text)`. Типы аргументов необходимы, если существует более чем одна функция с тем же именем.

Если функция не задана, для редактирования открывается пустой шаблон команды `CREATE FUNCTION`.

Если указан номер строки, psql будет позиционировать курсор на указанную строку тела функции. (Обратите внимание, что тело функции обычно не начинается на первой строке файла).

> **Tip:** См. ниже в разделе *Переменные окружения* о том, как настроить редактор.

`\encoding [` *`encoding`* `]`

Устанавливает кодировку набора символов на клиенте. Без аргумента, команда показывает текущую кодировку.

`\f [ string ]`

Устанавливает разделитель полей для невыровненного режима вывода запросов. По умолчанию используется вертикальная черта (`|`). См. также `\pset` для универсального способа настройки параметров вывода.

`\g [ filename ]`
`\g [ |command ]`

Отправляет текущий буфер запроса на сервер для выполнения, опционально сохраняет результат запроса в файле *`filename`* или перенаправляет вывод в команду оболочки ОС *`command`*. Вывод направляется в файл или команду, только если запрос успешно вернул 0 или более строк. Этого не происходит, если запрос завершился неудачно или выполнялась команда не возвращающая данные. `\g` без аргументов, по сути, эквивалентен точке с запятой. `\g` с аргументом является разовой альтернативой команде `\o`.

`\gset [ prefix ]`

Отправляет текущий буфер запроса на сервер для выполнения и сохраняет результат запроса в переменных psql (см. *Переменные*). Выполняемый запрос должен возвращать ровно одну строку. Каждый столбец строки результата сохраняется в отдельной переменной, которая называется также как и столбец. Например:

```
=> SELECT 'hello' AS var1, 10 AS var2
-> \gset
=> \echo :var1 :var2
hello 10
```

Если указан *`prefix`*, то он добавляется вначале к именам переменных:

```
=> SELECT 'hello' AS var1, 10 AS var2
-> \gset result_
=> \echo :result_var1 :result_var2
hello 10
```

Если значение столбца NULL, то вместо присвоения значения соответствующая переменная удаляется.

Если запрос завершается ошибкой или не возвращает одну строку, то никакие переменные не меняются.

`\h` или `\help [ command ]`

Выводит подсказку по синтаксису указанной команды SQL. Если *`command`* не указана, то psql выводит список всех команд, для которых доступна справка. Если в качестве command указана звёздочка (`*`), то выводится справка по всем командам SQL.

> **Note:** Для упрощения ввода, команды, состоящие из нескольких слов, можно не заключать в кавычки. Таким образом, можно просто писать `\help alter table`.

`\H` или `\html`

Включает вывод запросов в формате HTML. Если формат HTML уже включён, происходит переключение обратно на выровненный формат. Эта команда используется для совместимости и удобства, смотри `\pset` о том, как устанавливать другие параметры вывода.

`\i` или `\include` *`filename`*

Читает ввод из файла *`filename`* и выполняет его, как будто он был набран на клавиатуре.

> **Note:** Если вы хотите видеть строки файла на экране, по мере их чтения, необходимо установить переменную `ECHO` в значение `all`.

`\ir` или `\include_relative` *`filename`*

Команда `\ir` похож на `\i`, но по-разному интерпретирует относительные имена файлов. При выполнении в интерактивном режиме, две команды ведут себя одинаково. Однако при вызове из скрипта, `\ir` интерпретирует имена файлов относительно каталога, в котором расположен скрипт, а не текущего рабочего каталога.

`\l[+]` или `\list[+] [` *`pattern`* `]`

Выводит список баз данных на сервере и показывает их имена, владельцев, кодировку набора символов и права доступа. Если указан *`pattern`*, отображаются только базы данных, имена которых соответствуют шаблону. При добавлении `+` к имени команды, также отображаются: размер базы данных, табличное пространство по умолчанию и описание. (Информация о размере доступна только для баз данных, к которым текущий пользователь может подключиться.)

`\lo_export` *`loid filename`*

Читает большой объект с OID *`loid`* из базы данных и записывает его в файл *`filename`*. Обратите внимание, что это немного отличается от функции сервера `lo_export`, которая действует с правами пользователя, от имени которого работает сервер базы данных, и на файловой системе сервера.

> **Tip:** Используйте `\lo_list` для получения OID больших объектов.

`\lo_import` *`filename`* `[` *`comment`* `]`

Сохраняет файл в большом объекте PostgreSQL. Опционально, указанный комментарий связывается с объектом. Пример:

```
foo=> \lo_import '/home/peter/pictures/photo.xcf' 'a picture of me'
lo_import 152801
```

Ответ указывает на то, что большой объект получил OID 152801, который может быть

использован для доступа к вновь созданному объекту в будущем. Для удобства чтения, рекомендуется всегда связывать объекты с понятными комментариями. OID и комментарии можно посмотреть с помощью команды `\lo_list`.

Обратите внимание, что это немного отличается от функции сервера `lo_import`, т.к. действует от имени локального пользователя на локальной файловой системе, а не пользователя сервера на файловой системе сервера.

`\lo_list`

Показывает список всех больших объектов PostgreSQL, хранящихся в базе данных, вместе с предоставленными комментариями.

`\lo_unlink` *`loid`*

Удаляет большой объект с OID *`loid`* из базы данных.

Tip: Используйте `\lo_list` для получения OID больших объектов.

`\o` или `\out [` *`filename`* `]`
`\o` или `\out [ |`*`command`* `]`

Результаты запросов будут сохраняться в *`filename`* или перенаправляться в команду оболочки ОС *`command`*, Если аргумент не указан, вывод запросов переключается на стандартный вывод.

"Результаты запросов" включают в себя все таблицы, ответы команд, уведомления, полученные от сервера базы данных, а также вывод от мета-команд, запрашивающих базу данных (таких как `\d`), но не сообщения об ошибках.

Tip: Чтобы вставить текст между результатами запросов, используйте `\qecho`.

`\p` или `\print`

Печатает содержимое буфера запросов на стандартный вывод.

`\password [` *`username`* `]`

Изменяет пароль указанного пользователя (по умолчанию, текущего пользователя). Эта команда запрашивает новый пароль, шифрует и отправляет его на сервер в виде команды `ALTER ROLE`. Это гарантирует, что новый пароль не отображается в открытом виде в истории команд, журнале сервера или в другом месте.

`\prompt [` *`text`* `]` *`name`*

Предлагает пользователю ввести значение, которое будет присвоено переменной *`name`*. Опционально можно указать подсказку *`text`*. (Если подсказка состоит из нескольких слов, то её текст нужно взять в одинарные кавычки).

По умолчанию, `\prompt` использует терминал для ввода и вывода. Однако, если используется опция командной строки `-f`, `\prompt` использует стандартный ввод и стандартный вывод.

`\pset [ option [ value ] ]`

Эта команда устанавливает параметры, влияющие на вывод результатов запросов. *`option`* указывает, какой параметр необходимо установить. Семантика *`value`* меняется в зависимости от выбранного параметра. Для некоторых параметров, отсутствие *`value`* означает переключение значения, либо сброс значения, как описано ниже в разделе конкретного параметра. Если такое поведение не упоминается, то пропуск *`value`* приводит к отображению текущего значения параметра.

`\pset` без аргументов выводит текущий статус всех параметров команды.

Имеются следующие параметры:

`border`

`value` должно быть числом. В целом, чем выше число, тем больше границ и линий будут иметь таблицы, но это зависит от конкретного формата. В формате HTML это напрямую преобразуется в атрибут `border=...`. В других форматах имеют смысл только следующие значения: 0 (границ нет), 1 (разделительные линии внутри таблицы) и 2 (рамка вокруг таблицы). Форматы `latex` и `latex-longtable` также поддерживают значение 3, которое добавляет разделительные линии между строками.

`columns`

Устанавливает максимальную ширину для формата `wrapped`, а также ограничение по ширине свыше которого, будет требоваться пейджер для просмотра или переключение в вертикальное отображение при режиме `expanded auto`. При значении 0 (по умолчанию) максимальная ширина управляется переменной среды `COLUMNS` или шириной экрана, если `COLUMNS` не установлена. Кроме того, если `columns` равно нулю, то формат wrapped влияет только на вывод на экран. Если `columns` не равно 0, то это также влияет на вывод в файл или в другую команду через канал.

`expanded` (или `x`)

Для *`value`* возможны следующие значения: `on` или `off`, которые включают или отключают развернутый режим, или значение `auto`. Если *`value`* не указано, то команда переключает текущее значение в `on` или `off`. Если включен развернутый режим, результаты запроса отображаются в две колонки: имя столбца в левой колонке, данные в правой. Этот режим полезен, если данные не помещаются на экране в обычном "горизонтальном" режиме. При выборе `auto`, расширенный режим используется всякий раз, когда результат запроса шире, чем экран, в противном случае используется обычный режим. `auto` действует только в

форматах aligned и wrapped. В других форматах, поведение такое, как если расширенный режим отключен.

fieldsep

Устанавливает разделитель полей для невыровненного режима вывода запросов. Таким образом, можно формировать вывод, в котором значения будут разделены табуляцией или запятыми. Это может быть предпочтительным для использования в других программах. Для установки символа табуляции в качестве разделителя полей, введите \pset fieldsep '\t'. По умолчанию используется вертикальная черта ('|').

fieldsep_zero

Устанавливает разделитель полей для невыровненного режима вывода в нулевой байт.

footer

Для *value* возможны два значения: on или off, которые включают или отключают вывод результирующей строки с количеством выбранных записей (*n* строк). Если *value* не указано, то команда переключает текущее значение в on или off.

format

Устанавливает формат вывода в один из следующих: unaligned, aligned, wrapped, html, latex (использует tabular), latex-longtable или troff-ms. Допускается сокращение слова до уникального значения. (Это значит, что одной буквы будет достаточно.)

В unaligned формате все столбцы размещаются на одной строке и отделяются друг от друга разделителем полей. Это полезно для создания вывода, который будет читаться другими программами (например, формат с выводом значений разделенных запятыми или табуляцией).

Формат aligned это стандартный, удобочитаемый, хорошо отформатированный текстовый вывод. Используется по умолчанию.

Формат wrapped похож на aligned, но переносит длинные значения столбцов на новые строки, чтобы общий вывод поместился в заданную ширину. Задание ширины вывода описано в параметре columns. Обратите внимание, что psql не будет пытаться переносить на новые строки заголовки столбцов. Поэтому формат wrapped работает так же, как aligned если общая ширина, требуемая для всех заголовков столбцов, превышает установленную максимальную ширину.

Форматы html, latex, latex-longtable и troff-ms выводят таблицы, которые предназначены для включения в документы с помощью соответствующего языка разметки. Они не являются полноценными документами!

Возможно это не обязательно в HTML, но в LaTeX необходимо иметь полный упаковщик документа. `latex-longtable` также требует установленных LaTeX пакетов `longtable` и `booktabs`.

`linestyle`

Задает стиль отрисовки линий границы в одно из значений: `ascii`, `old-ascii` или `unicode`. Допускается сокращение слова до уникального значения. (Это значит, что одной буквы будет достаточно.) Значение по умолчанию: `ascii`. Этот параметр действует только в форматах `aligned` и `wrapped`.

Стиль `ascii` использует обычные символы ASCII. Символы новой строки в данных показываются с использованием символа `+` в правом поле. Когда при формате `wrapped` происходит перенос данных на новую строку (без символа новой строки), ставится точка (`.`) в правом поле первой строки и точка в левом поле следующей строки.

Стиль `old-ascii` использует обычные символы ASCII, в стиле PostgreSQL 8.4 и раньше. Символы новой строки в данных отображаются используя символ `:` вместо левого разделителя полей. Когда происходит перенос данных на новую строку без символа новой строки, символ `;` используется вместо левого разделителя полей.

Стиль `unicode` использует символы Юникод для рисования линий. Символы новой строки в данных показываются с использованием символа возврата каретки в правом поле. Когда при формате `wrapped` происходит перенос данных на новую строку (без символа новой строки), ставится символ многоточия в правом поле первой строки и в левом поле следующей строки.

Когда `border` больше нуля, то параметр также определяет символы, которыми будут рисоваться границы. Обычные символы ASCII будут работать везде, но символы в Юникоде выглядят лучше на дисплеях, распознающих их.

`null`

Устанавливает строку, которая будет напечатана вместо значения null. По умолчанию не печатается ничего, что можно ошибочно принять за пустую строку. Например, можно было бы предпочесть `\pset null '(null)'`.

`numericlocale`

Для *`value`* возможны два значения: `on` или `off`, которые включают или отключают вывод чисел в локализованном формате. Если *`value`* не указано, то команда переключает вывод чисел с локализованного на обычный и обратно.

`pager`

Контролирует использование пейджера для просмотра результатов запросов и справочной информации psql. Если переменная среды PAGER установлена, то

данные передаются в указанную программу. В противном случае используется платформозависимая программа по умолчанию (например, `more`).

Если `pager` имеет значение `off`, программа пейджер не используется. Если `pager` имеет значение `on`, программа пейджер используется при необходимости, т.е. когда вывод на терминал не помещается на экране. Параметр pager также может быть установлен в значение `always`, при этом программа пейджер будет использоваться всегда, независимо от того, помещается вывод на терминал на экран или нет. `\pset pager` без указания *`value`* переключает значения `on` и `off`.

`recordsep`

Устанавливает разделитель записей (строк) для невыровненного режима вывода. По умолчанию используется символ новой строки.

`recordsep_zero`

Устанавливает разделитель записей для невыровненного режима вывода в нулевой байт.

`tableattr` (или `T`)

Устанавливает атрибуты, которые будут помещены в тег `table`, при формате вывода HTML. Например, это может быть `cellpadding` или `border`. Заметьте, что, вероятно, не нужно здесь задавать `border`, т.к. для этого уже есть `\pset border`. Если *`value`* не задано, атрибуты таблицы удаляются.

В формате `latex-longtable`, этот параметр контролирует пропорциональную ширину каждого столбца, данные которого выравнены по левому краю. Он указывается как список разделенных пробелами значений, например `'0.2 0.2 0.6'`. Для неуказанных столбцов используется последнее из заданных значений.

`title`

Устанавливает заголовок таблицы для любых впоследствии выводимых таблиц. Это можно использовать для задания описательных тегов при формировании вывода. Если *`value`* не задано, заголовок таблицы удаляется.

`tuples_only` (или `t`)

Для *`value`* возможны два значения: `on` или `off`, которые включают или отключают режим вывода только кортежей. Если *`value`* не указано, то команда переключает с режима вывода только кортежей на обычный режим и обратно. Обычный вывод включает в себя дополнительную информацию, такую как заголовки столбцов и различные колонтитулы. В режиме вывода только кортежей отображаются только фактические табличные данные.

Иллюстрацию того, как могут выглядеть различные форматы можно посмотреть в

разделе *Примеры*.

> **Tip:** Для некоторых параметров `\pset` есть короткие команды. См. `\a`, `\C`, `\H`, `\t`, `\T` и `\x`.

`\q` или `\quit`

Выход из psql. При использовании в скрипте, прекращается только выполнение этого скрипта.

`\qecho` *`text`* `[ ... ]`

Эта команда идентична `\echo` за исключением того, что вывод будет записываться в канал вывода запросов, установленный `\o`.

`\r` или `\reset`

Сбрасывает (очищает) буфер запроса.

`\s [` *`filename`* `]`

Записывает историю команд psql в файл *`filename`*. Если *`filename`* не указан, то история команд выводится на стандартный вывод (с использованием пейджера, где уместно). Эта опция недоступна, если psql был собран без поддержки Readline.

`\set [` *`name`* `[` *`value`* `[ ... ] ] ]`

Задаёт psql переменной *`name`* значение *`value`* или, если задано более одного значения, их конкатенацию. Если задан только один аргумент, то переменной с этим именем присваивается пустое значение. Для удаления переменной, используйте команду `\unset`.

`\set` без аргументов выводит имена и значения всех psql переменных, установленных в настоящее время.

Имена переменных могут содержать буквы, цифры и знаки подчеркивания. См. *Переменные* ниже для деталей. Имена переменных чувствительны к регистру.

Хотя вы можете задать любой переменной любое значение, psql рассматривает несколько переменных особым образом. Они документированы в разделе о переменных.

> **Note:** Эта команда не имеет отношения к SQL команде SET.

`\setenv` *`name`* `[` *`value`* `]`

Задает переменной среды *`name`* значение *`value`*, или, если *`value`* не задано, удаляет переменную среды. Пример:

```
testdb=> \setenv PAGER less
testdb=> \setenv LESS -imx4F
```

`\sf[+]` *`function_description`*

Извлекает из базы данных и выводит определение заданной функции, в форме команды `CREATE OR REPLACE FUNCTION`. Определение печатается в текущий канал вывода запросов, установленный `\o`.

Для функции может быть задано только имя или имя и аргументы, например `foo(integer, text)`. Типы аргументов необходимы, если существует более чем одна функция с тем же именем.

При добавлении `+` к имени команды строки вывода нумеруются, первая строка тела функции получит номер 1.

`\t`

Включает/выключает отображение имен столбцов и результирующей строки с количеством выбранных записей для запросов. Эта команда эквивалентна `\pset tuples_only` и предоставлена для удобства.

`\T` *`table_options`*

Устанавливает атрибуты, которые будут помещены в тег `table`, при HTML формате вывода. Эта команда эквивалентна `\pset tableattr` *`table_options`*.

`\timing [` *`on`* `|` *`off`* `]`

Включает/отключает отображение времени выполнения команд SQL, в миллисекундах. Без параметра переключает текущий режим отображения времени выполнения.

`\unset` *`name`*

Удаляет psql переменную *`name`*.

`\w` или `\write` *`filename`*
`\w` или `\write |`*`command`*

Выводит текущий буфер запроса в файл *`filename`* или передает в команду ОС *`command`*.

`\watch [` *`seconds`* `]`

Команда многократно выполняет текущий буфер запроса (подобно `\g`), пока не будет прервана или не произойдет сбой запроса. Аргумент задает количество секунд ожидания между выполнениями запроса (по умолчанию 2 секунды).

`\x [` *`on`* `|` *`off`* `|` *`auto`* `]`

Устанавливает или переключает режим развернутого вывода таблицы. Это эквивалентно `\pset expanded`.

`\z [ pattern ]`

Выводит список таблиц, представлений и последовательностей с их правами доступа. Если указан *pattern*, отображаются только таблицы, представления и последовательности, имена которых соответствуют шаблону.

Это псевдоним для `\dp` ("показать права доступа").

`\! [ command ]`

Выполняет команду ОС *command*. Без указания *command* запускает отдельную командную оболочку ОС. Последующие аргументы не интерпретируются, командная оболочка ОС увидит их как есть. В частности, не применяются правила подстановки переменных и экранирование при помощи бэкслеш.

`\?`

Показывает справочную информацию о мета-командах.

Шаблоны поиска (Patterns)

Различные `\d` команды принимают параметр *pattern* для указания имени(имен) объектов для отображения. В простейшем случае шаблон - это точное имя объекта. Символы внутри шаблона обычно приводятся к нижнему регистру, как и для имен SQL объектов; к примеру `\dt FOO` выводит таблицу с именем `foo`. Как и для SQL имен, двойные кавычки вокруг шаблона предотвращают перевод в нижний регистр. Для включения символа двойной кавычки в шаблон, используются два символа двойных кавычек подряд внутри шаблона в двойных кавычках. Опять же, это соответствует правилам для SQL идентификаторов. Например `\dt "FOO""BAR"` будет выводить таблицу с именем `FOO"BAR` (но не `foo"bar`). В отличие от обычных правил для SQL имен, можно взять в двойные кавычки только часть шаблона, например `\dt FOO"FOO"BAR` будет выводить таблицу с именем `fooFOObar`.

Если *pattern* вообще не указан, `\d` команды выводят все объекты, видимые для текущей схемы. Это эквивалентно указанию `*` в качестве шаблона. (Объект считается *видимым*, если схема, в которой он находится, лежит на пути поиска, и объект с таким типом и именем на пути поиска еще не появлялся. Это эквивалентно утверждению, что на объект можно ссылаться по имени, без явного указания схемы.) Чтобы увидеть все объекты в базе данных, независимо от видимости, в качестве шаблона используется `*.*`.

Внутри шаблона `*` обозначает любое количество символов, включая отсутствие символов. `?` соответствует любому одному символу. (Это соответствует шаблонам имен файлов в Unix.) Например, `\dt int*` отображает все таблицы, чьи имена начинаются на `int`. Однако внутри двойных кавычек `*` и `?` теряют своё специальное значение и становятся обычными

символами.

Шаблон, содержащий точку (`.`), интерпретируется как шаблон имени схемы, за которым следует шаблон имени объекта. Например, `\dt foo*.*bar*` отображает все таблицы, имена которых включают `bar`, и расположенные в схемах, имена которых начинаются с `foo`. Шаблону, не содержащему точку, могут соответствовать только объекты текущей схемы. Опять же, точка внутри двойных кавычек теряет своё специальное значение.

Опытные пользователи могут использовать возможности регулярных выражений, такие как классы символов. Например `[0-9]` соответствует любой цифре. Все специальные символы регулярных выражений работают как описано в Section 9.7.3, за исключением: `.` используется в качестве разделителя как говорилось выше; `*` соответствует регулярному выражению `.*`; `?` соответствует `.` , а также символ `$` который не имеет специального значения. При необходимости эти символы можно эмулировать указывая `?` для эмуляции `.`, `(`*R*`+|)` для *R*`*`, `(`*R*`|)` для *R*`?`. `$` не требуется, как символ регулярного выражения, потому что шаблон должен соответствовать имени целиком, в отличие от обычной интерпретации регулярных выражений (другими словами, `$` автоматически добавляется в шаблон). Используйте `*` в начале и/или в конце, если не хотите, чтобы шаблон закреплялся. Обратите внимание, что внутри двойных кавычек, все специальные символы регулярных выражений теряют свое специальное значение и соответствуют сами себе. Также, специальные символы регулярных выражений не действуют в шаблонах для имен операторов (т.е. в аргументе команды `\do`).

Расширенные возможности

Переменные

psql предоставляет возможности подстановки переменных подобные тем, что используются в командных оболочках Unix. Переменные представляют собой пары имя/значение, где значением может быть любая строка любой длины. Имя должно состоять из букв (включая нелатинские буквы), цифр и знаков подчеркивания.

Чтобы установить переменную, используется psql мета-команда `\set`. Например:

```
testdb=> \set foo bar
```

присваивает переменной `foo` значение `bar`. Чтобы получить значение переменной, нужно поставить двоеточие перед её именем, например:

```
testdb=> \echo :foo
bar
```

Это работает как в обычных SQL командах, так и в мета-командах, подробности в разделе *Интерполяция SQL* ниже.

При вызове `\set` без второго аргумента, переменной присваивается пустая строка. Для удаления переменной, используется команда `\unset`. Чтобы посмотреть значения всех

переменных, нужно вызвать `\set` без аргументов.

> **Note:** На аргументы `\set` распространяются те же правила подстановки, что и для других команд. Таким образом можно создавать интересные ссылки, например `\set :foo 'something'` получая "мягкие ссылки" в Perl или "переменные переменных" в PHP. К сожалению (или к счастью?), с этими конструкциями нельзя сделать ничего полезного. С другой стороны `\set bar :foo` является прекрасным способом копирования переменной.

Некоторые переменные, обрабатываются в psql особым образом. Они представляют собой определенные параметры, которые могут быть изменены во время выполнения, путем присваивания нового значения, а в некоторых переменных содержится изменяемое состояние psql. Хотя эти переменные можно использовать и для других целей, делать это не рекомендуется, так как поведение программы может быстро стать очень странным. По соглашению, имена специальных переменных состоят из всех заглавных ASCII букв (и, возможно, цифр и знаков подчеркивания). Для максимальной совместимости в будущем, старайтесь не использовать такие имена для собственных переменных. Далее идет список переменных, обрабатываемых особым образом.

`AUTOCOMMIT`

> При значении `on` (по умолчанию) после каждой успешно выполненной команды выполняется фиксация изменений. Чтобы отложить фиксацию изменений в этом режиме, нужно выполнить SQL команду `BEGIN` или `START TRANSACTION`. При значении `off` или если переменная не определена, фиксация изменений не происходит до тех пор, пока явно не выполнена команда `COMMIT` или `END`. При значении `off` неявно выполняется `BEGIN`, непосредственно перед любой командой, за исключением случаев когда: команда уже в транзакционном блоке; перед самой командой `BEGIN` или другой командой управления транзакциями; перед командой, которая не может выполняться внутри транзакционного блока (например `VACUUM`).
>
> > **Note:** Если режим `autocommit` отключен, необходимо явно откатывать изменения в неуспешных транзакциях, выполняя команду `ABORT` или `ROLLBACK`. Также имейте в виду, что при выходе из сессии без фиксации изменений, несохраненные изменения будут потеряны.
> >
> > **Note:** Включенный режим `autocommit` является традиционным для PostgreSQL, а выключенный режим ближе к спецификации SQL. Если вы предпочитаете отключить режим `autocommit`, это можно сделать в общесистемном файле `psqlrc` или в персональном файле `~/.psqlrc`.

`COMP_KEYWORD_CASE`

> Определяет, какой регистр букв будет использован при автоматическом завершении ключевых слов SQL. Если установлено в `lower` или `upper`, будет использоваться нижний или верхний регистр, соответственно. Если установлено в `preserve-lower` или `preserve-upper` (по умолчанию), то завершаемое слово будет в том же регистре, что и уже введенное начало слова, но последующие слова, завершаемые полностью,

будут в нижнем или верхнем регистре, соответственно.

`DBNAME`

Имя базы данных, к которой вы сейчас подключены. Устанавливается всякий раз при подключении к базе данных (в том числе при старте программы), но эту переменную можно удалить.

`ECHO`

Если установлена в `all`, все непустые входящие строки выводятся на стандартный вывод по мере их чтения. (Это не относится к интерактивному режиму.) Чтобы включить такое поведение при старте psql, используйте опцию `-a`. Если установлена в `queries`, psql выводит каждый запрос на стандартный вывод после отправки его на сервер. Опция командной строки для такого поведения `-e`.

`ECHO_HIDDEN`

Если эта переменная установлена в значение `on`, то при выполнении мета-команд, запрашивающих базу данных, сначала будет выводиться текст запроса. Это позволяет изучать внутреннее устройство PostgreSQL и реализовывать похожую функциональность в своих программах. (Чтобы включить такое поведение при старте psql, используете опцию `-E`.) При установке переменной в значение `noexec`, запросы просто отображаются, но не отправляются на сервер и не выполняются.

`ENCODING`

Содержит текущую кодировку набора символов клиента.

`FETCH_COUNT`

Если переменная установлена в целое значение > 0, результаты запросов `SELECT` извлекаются из базы данных и отображаются группами с таким количеством строк, в отличии от поведения по умолчанию, когда перед отображением результирующий набор накапливается целиком. Это позволяет использовать ограниченный размер памяти, независимо от размера выборки. При включении этой функциональности обычно используются значения от 100 до 1000. Имейте в виду, что запрос может завершиться ошибкой после отображения некоторого количества строк.

> **Tip:** Хотя можно использовать любой формат вывода, формат по умолчанию `aligned` как правило выглядит хуже, потому что каждая группа по `FETCH_COUNT` строк форматируется отдельно, что может привести к разной ширине столбцов в разных группах. Остальные форматы вывода работают лучше.

`HISTCONTROL`

Если переменная установлена в `ignorespace`, строки начинающиеся с пробела не сохраняются в истории. Если установлена в `ignoredups`, строки, которые уже есть в

истории, повторно не сохраняются. Значение `ignoreboth` сочетает в себе оба варианта. Если значение не задано или отличается от перечисленных выше, все строки, введенные в интерактивном режиме, сохраняются в истории.

Note: Эта функциональность была бессовестно списана с Bash.

`HISTFILE`

Имя файла для хранения истории команд. Значение по умолчанию `~/.psql_history`. Например, установив:

```
\set HISTFILE ~/.psql_history- :DBNAME
```

в `~/.psqlrc` psql будет поддерживать отдельную историю для каждой базы данных.

Note: Эта функциональность была бессовестно списана с Bash.

`HISTSIZE`

Количество команд для хранения в истории. Значение по умолчанию 500.

Note: Эта функциональность была бессовестно списана с Bash.

`HOST`

Содержит имя хоста, где расположен сервер базы данных, к которому вы сейчас подключены. Устанавливается всякий раз при подключении к базе данных (в том числе при старте программы), но эту переменную можно удалить.

`IGNOREEOF`

Если не установлена, то ввод символа EOF (обычно **Control**+**D**) в интерактивной сессии psql завершает работу приложения. Если задано числовое значение, то именно такое количество символов EOF будет проигнорировано, прежде чем приложение завершит работу. Если переменная задана, но имеет не числовое значение, то значение по умолчанию 10.

Note: Эта функциональность была бессовестно списана с Bash.

`LASTOID`

Содержит значение последнего OID, полученного командой `INSERT` или `\lo_import`. Корректное значение переменной гарантируется до тех пор, пока не будет отображен результат следующей SQL команды.

`ON_ERROR_ROLLBACK`

При значении `on`, если команда в транзакционном блоке выдает ошибку, ошибка игнорируется и транзакция продолжается. При значении `interactive` такие ошибки игнорируются только в интерактивных сессиях, но не в скриптах. Если переменная не

установлена или при значении `off`, команда вызвавшая ошибку в транзакционном блоке, отменяет всю транзакцию. Режим отката транзакции при возникновении ошибки обеспечивается выполнением неявных команд `SAVEPOINT` перед каждой командой в транзакционном блоке и откатом к точке сохранения в случае ошибки.

`ON_ERROR_STOP`

По умолчанию, после возникновения ошибки обработка команд продолжается. Если эта переменная установлена в значение `on`, обработка команд будет немедленно прекращена. В интерактивном режиме psql вернется в командную строку; иначе psql прекратит работу с кодом возврата 3, чтобы отличить этот случай от фатальных ошибок, для которых используется код возврата 1. В любом случае, выполнение всех запущенных скриптов (верхнеуровневый скрипт и любые другие, которые он мог запустить) будет немедленно прекращено. Если верхнеуровневая командная строка содержит несколько SQL команд, выполнение завершится на текущей команде.

`PORT`

Содержит порт сервера базы данных, к которому вы сейчас подключены. Устанавливается всякий раз при подключении к базе данных (в том числе при старте программы), но эту переменную можно удалить.

`PROMPT1`
`PROMPT2`
`PROMPT3`

Указывают, как должны выглядеть приглашения psql. См. *Настройка приглашений* ниже.

`QUIET`

Установка значения `on` эквивалента опции командной строки `-q`. Это, вероятно, не слишком полезно в интерактивном режиме.

`SINGLELINE`

Установка значения `on` эквивалента опции командной строки `-S`.

`SINGLESTEP`

Эта переменная эквивалента опции командной строки `-s`.

`USER`

Содержит имя пользователя базы данных, который сейчас подключен. Устанавливается всякий раз при подключении к базе данных (в том числе при старте программы), но эту переменную можно удалить.

`VERBOSITY`

Этой переменной можно присвоить значения `default`, `verbose` или `terse` для контроля уровня детализации отчетов об ошибках.

Интерполяция SQL

Ключевой особенностью переменных psql является возможность подставлять ("интерполировать") их в команды SQL, также как и в аргументы мета-команд. Кроме того, psql предоставляет средства для корректного использования кавычек для значений переменных, которые используются как SQL литералы или идентификаторы. Синтаксис для подстановки значения без кавычек - это добавить перед именем переменной двоеточие (`:`). Например:

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :foo;
```

будет запрашивать таблицу `my_table`. Обратите внимание, что это может быть небезопасным: значение переменной копируется буквально, поэтому оно может содержать непарные кавычки, или даже мета-команды. При применении необходимо убедиться, что это имеет смысл.

Когда значение будет использоваться в качестве SQL литерала или идентификатора, безопаснее заключить его в кавычки. Если значение переменной используется как SQL литерал, то после двоеточия нужно написать имя переменной в одинарных кавычках. Если значение переменной используется как SQL идентификатор, то после двоеточия нужно написать имя переменной в двойных кавычках. Эти конструкции корректно работают с кавычками и другими специальными символами, которые могут содержаться в значении переменной. Предыдущий пример более безопасно выглядит так:

```
testdb=> \set foo 'my_table'
testdb=> SELECT * FROM :"foo";
```

Подстановка переменных не будет выполняться, если SQL литералы или идентификаторы заключены в кавычки. Поэтому конструкция `':foo'` не превратится во взятое в кавычки значение переменной (и это было бы не безопасно, если бы работало, т.к. обработка кавычек внутри значения переменной была бы некорректной).

Один из примеров использования данного механизма - это копирование содержимого файла в столбец таблицы. Сначала загрузим содержимое файла в переменную, затем подставим значение переменной как строку в кавычках:

```
testdb=> \set content `cat my_file.txt`
testdb=> INSERT INTO my_table VALUES (:'content');
```

(Отметим, что это пока не будет работать, если `my_file.txt` содержит байт NUL. psql не поддерживает NUL в значениях переменных.)

Так как двоеточие может легально присутствовать в SQL командах, попытка подстановки (например для `:name`, `:'name'` или `:"name"`) не выполняется, если переменная не

установлена. В любом случае, можно экранировать двоеточие с помощью обратной косой черты, чтобы предотвратить подстановку.

Использование двоеточия для переменных является стандартом SQL для встраиваемых языков запросов, таких как ECPG. Использование двоеточия для срезов массивов и приведения типов является расширениями PostgreSQL, что иногда может конфликтовать со стандартным использованием. Использование двоеточия и кавычек для экранирования значения переменной при подстановке в качестве SQL литерала или идентификатора - это расширение psql.

Настройка приглашений

Приглашения psql можно настроить по своему предпочтению. Три переменные `PROMPT1`, `PROMPT2`, и `PROMPT3` содержат строки и специальные escape-последовательности, которые описывают внешний вид приглашения. `PROMPT1` это обычное приглашение, которое выдается, когда psql ожидает ввода новой команды. `PROMPT2` выдается при переходе на новую строку, когда вводимая команда не завершается точкой с запятой или осталась не закрытая кавычка. `PROMPT3` выдается при выполнении SQL команды `COPY`, когда ожидается ввод значений строк с терминала.

Значения этих переменных выводятся буквально, за исключением случаев, когда в них встречается знак процента (%). В зависимости от следующего символа, будет подставляться определенный текст. Существуют следующие подстановки:

`%M`

Полное имя хоста (с доменным именем) сервера базы данных, или `[local]` если подключение выполнено через сокет Unix домена, или `[local:`*`/dir/name`*`]`, если сокет Unix домена не скомпилирован в местоположение по умолчанию.

`%m`

Имя хоста сервера базы данных, усеченное до первой точки или `[local]` если подключение выполнено через сокет Unix домена.

`%>`

Номер порта, который прослушивает сервер базы данных.

`%n`

Имя пользователя базы данных для текущей сессии. (Это значение может меняться в течении сессии в результате выполнения команды `SET SESSION AUTHORIZATION`.)

`%/`

Имя текущей базы данных.

%~

Похоже на %/, но выводит тильду ~, если текущая база данных совпадает с базой данных по умолчанию.

%#

Если пользователь текущей сессии является суперпользователем базы данных, то выводит #, иначе >. (Это значение может меняться в течении сессии в результате выполнения команды SET SESSION AUTHORIZATION.)

%R

Для PROMPT1 это обычно соответствует =, но в однострочном режиме выводит ^, а если произошло отключение от базы данных (например при сбое в \connect) то !. Для PROMPT2 будет выводиться -, *, одинарная кавычка, двойная кавычка или знак доллара в зависимости от того, почему потребовалась дополнительная строка ввода: команда не была завершена, мы находимся внутри комментария /* ... */ или потому, что мы внутри кавычек или внутри строки экранированной знаками доллара. Для PROMPT3 эта последовательность ничего не выводит.

%x

Состояние транзакции: пустая строка если не в транзакционном блоке; * когда в транзакционном блоке; ! когда в транзакционном блоке, в котором произошла ошибка и ? когда состояние транзакции не определено (например, нет подключения к базе данных).

%*digits*

Подставляется символ с указанным восьмеричным кодом.

%:*name*:

Подставляется значение psql переменной *name*. См. *Переменные* для деталей.

%`*command*`

Подставляется вывод команды *command*, как и в обычной подстановке с обратными апострофами.

%[... %]

Приглашения могут содержать управляющие символы терминала, которые, например, изменяют цвет, фон и стиль текста приглашения или изменяют заголовок окна терминала. Для того, чтобы возможности редактирования Readline работали правильно, непечатаемые символы нужно расположить между %[и %], чтобы сделать невидимыми. Можно делать несколько таких включений в приглашение. Например:

```
testdb=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%# '
```

выдаст жирное (`1;`), желтое на черном (`33;40`) приглашение для VT100 совместимых цветных терминалов.

Чтобы вставить знак процента нужно написать %%. По умолчанию используются значения `'%/%R%# '` для `PROMPT1` и `PROMPT2`, и `'>> '` для `PROMPT3`.

Note: Эта функциональность была бессовестно списана с tcsh.

Редактирование командной строки

psql поддерживает библиотеку Readline для удобного редактирования командной строки. История команд автоматически сохраняется при выходе из psql и загружается при запуске. Завершение клавишей `TAB` также поддерживается, хотя логика завершения не претендует на роль парсера SQL. Запросы, генерируемые завершением по `TAB`, также могут конфликтовать с другими командами SQL, например `SET TRANSACTION ISOLATION LEVEL`. Если по какой-либо причине вам не нравится завершение по клавише `TAB`, его можно отключить в файле `.inputrc` в вашей домашней директории:

```
$if psql
set disable-completion on
$endif
```

(Это возможность не psql, а Readline. Читайте документацию к Readline для дополнительной информации.)

Переменные окружения

`COLUMNS`

Если `\pset columns` равно нулю, управляет шириной формата вывода `wrapped`, а также определяет нужно ли использовать пейджер и нужно ли переключаться в вертикальный формат в режиме `expanded auto`.

`PAGER`

Если результат запроса не помещается на экране, он передается в эту команду для отображения. Обычно это `more` или `less`. Значение по умолчанию зависит от платформы. Использование пейджера можно отключить с помощью команды `\pset`.

`PGDATABASE`
`PGHOST`
`PGPORT`
`PGUSER`

Параметры подключения по умолчанию (см. Section 31.14).

`PSQL_EDITOR`

`EDITOR`
`VISUAL`

Редактор, используемый командами `\e` и `\ef`. Переменные рассматриваются в перечисленном порядке. Будет использовано значение первой установленной переменной.

По умолчанию используются `vi` на Unix системах и `notepad.exe` на Windows.

`PSQL_EDITOR_LINENUMBER_ARG`

Если в командах `\e` или `\ef` указан номер строки, эта переменная задает название опции командной строки, которая используется для передачи номера строки в редактор. Для таких редакторов как vi или Emacs это знак плюс. Добавьте в конец значения пробел, если он требуется для разделения названия опции и номера строки. Примеры:

```
PSQL_EDITOR_LINENUMBER_ARG='+'
PSQL_EDITOR_LINENUMBER_ARG='--line '
```

Значение по умолчанию `+` на Unix системах (соответствует редактору по умолчанию `vi` и многим другим распространенным редакторам). На платформе Windows нет значения по умолчанию.

`PSQL_HISTORY`

Альтернативное расположение файла с историей команд. Допускается использование тильды (`~`).

`PSQLRC`

Альтернативное расположение пользовательского файла `.psqlrc`. Допускается использование тильды (`~`).

`SHELL`

Команда операционной системы, выполняемая мета-командой `\!`.

`TMPDIR`

Каталог для хранения временных файлов. По умолчанию `/tmp`.

Эта утилита, как и большинство других утилит PostgreSQL, также использует переменные среды, поддерживаемые libpq (см. Section 31.14).

Файлы

`psqlrc` и `~/.psqlrc`

При запуске без опций `-X` или `-c` psql пытается считать и выполнить команды из

общесистемного стартового файла (psqlrc), а затем из персонального стартового файла пользователя (~/.psqlrc). Это происходит после подключения к базе данных, но до получения обычных команд. Эти файлы могут использоваться для настройки клиента и/или сервера, как правило, с помощью команд \set и SET.

Общесистемный стартовый файл называется psqlrc, он будет искаться в каталоге установке "конфигурация системы". Для того чтобы узнать этот каталог, надежнее всего выполнить команду pg_config --sysconfdir. По умолчанию он расположен в ../etc/ относительно каталога, содержащего исполняемые файлы PostgreSQL. Имя этого каталога можно задать явно через переменную окружения PGSYSCONFDIR.

Персональный стартовый файл пользователя называется .psqlrc, он будет искаться в домашнем каталоге вызывающего пользователя. В Windows, где отсутствует такое понятие, персональный стартовый файл называется %APPDATA %\postgresql\psqlrc.conf. Расположение персонального стартового файла пользователя можно задать явно через переменную окружения PSQLRC.

Оба стартовых файла общесистемный и персональный можно привязать к конкретной версии psql. Для этого в конце имени файла нужно добавить номер мажорного или минорного релиза PostgreSQL, например ~/.psqlrc-9.2 или ~/.psqlrc-9.2.5. При наличии нескольких файлов, файл с более детальным номером версии будет иметь предпочтение.

.psql_history

История командной строки хранится в файле ~/.psql_history или %APPDATA %\postgresql\psql_history на Windows.

Расположение файла истории можно задать явно через переменную окружения PSQL_HISTORY.

Замечания

- В ранних версиях psql первый аргумент в однобуквенных мета-командах мог начинаться сразу после самой команды, без промежуточных пробелов. Начиная с PostgreSQL 8.4 это не разрешается.
- psql лучше всего работает с серверами той же или более старой мажорной версии. Сбой мета-команды наиболее вероятен, если версия сервера новее, чем версия psql. Однако, мета-команды семейства \d должны работать с версиями сервера до 7.4, хотя и необязательно с серверами новее, чем сам psql. Общая функциональность запуска SQL команд и отображения результатов запросов также должна работать на серверах с более новой мажорной версией, но это не гарантируется во всех случаях.

  Если вы хотите подключаться к нескольким серверам с различными мажорными версиями, рекомендуется использовать последнюю версию psql. Кроме того можно сохранить копию psql от каждой мажорной версии и использовать ту, которая

соответствует версии сервера. Но на практике, в этих дополнительных сложностях нет необходимости.

Замечания для пользователей Windows

psql создан как "консольное приложение". Поскольку в Windows консольные окна используют кодировку символов отличную от той, что используется для остальной системы, нужно проявить особую осторожность при использовании 8-битных символов. Если psql обнаружит проблемную кодовую страницу консоли, он предупредит вас при запуске. Чтобы изменить кодовую страницу консоли, необходимы две вещи:

- Задать кодовую страницу, выполнив `cmd.exe /c chcp 1251`. (1251 это кодовая страница для России, замените на ваше значение.) При использовании Cygwin, эту команду можно записать в `/etc/profile`.
- Установите консольный шрифт в `Lucida Console`, потому что растровый шрифт не работает с кодовой страницей ANSI.

Примеры

Первый пример показывает, что для ввода одной команды может потребоваться несколько строк. Обратите внимание, как меняется приглашение:

```
testdb=> CREATE TABLE my_table (
testdb(>  first integer not null default 0,
testdb(>  second text)
testdb-> ;
CREATE TABLE
```

Теперь посмотрим на определение таблицы:

```
testdb=> \d my_table
             Таблица "my_table"
 Колонка |  Тип    |    Модификаторы
---------+---------+--------------------
 first   | integer | NOT NULL DEFAULT 0
 second  | text    |
```

Теперь изменим приглашение на что-то более интересное:

```
testdb=> \set PROMPT1 '%n@%m %~%R%# '
peter@localhost testdb=>
```

Предположим, что вы внесли данные в таблицу и хотите на них посмотреть:

```
peter@localhost testdb=> SELECT * FROM my_table;
 first | second
-------+--------
     1 | Один
     2 | Два
     3 | Три
     4 | Четыре
(4 строки)
```

Таблицу можно вывести разными способами при помощи команды `\pset`:

```
peter@localhost testdb=> \pset border 2
Установлен стиль границ: 2.
peter@localhost testdb=> SELECT * FROM my_table;
+-------+--------+
| first | second |
+-------+--------+
|     1 | Один   |
|     2 | Два    |
|     3 | Три    |
|     4 | Четыре |
+-------+--------+
(4 строки)

peter@localhost testdb=> \pset border 0
Установлен стиль границ: 0.
peter@localhost testdb=> SELECT * FROM my_table;
first second
----- ------
    1 Один
    2 Два
    3 Три
    4 Четыре
(4 строки)

peter@localhost testdb=> \pset border 1
Установлен стиль границ: 1.
peter@localhost testdb=> \pset format unaligned
Формат вывода: unaligned.
peter@localhost testdb=> \pset fieldsep ','
Разделитель полей: ",".
peter@localhost testdb=> \pset tuples_only
Выводятся только кортежи.
peter@localhost testdb=> SELECT second, first FROM my_table;
Один,1
Два,2
Три,3
Четыре,4
```

Альтернативно, можно использовать короткие команды:

```
peter@localhost testdb=> \a \t \x
Формат вывода: aligned.
Режим вывода только кортежей выключен.
Расширенный вывод включен.
peter@localhost testdb=> SELECT * FROM my_table;
-[ RECORD 1 ]-
first  | 1
second | Один
-[ RECORD 2 ]-
first  | 2
second | Два
-[ RECORD 3 ]-
first  | 3
second | Три
-[ RECORD 4 ]-
first  | 4
second | Четыре
```