psql

psql — terminal interactif PostgreSQL™

Synopsis

`psql` [*option*...] [*nombase* [*nomutilisateur*]]

Description

psql est une interface en mode texte pour PostgreSQL™. Il vous permet de saisir des requêtes de façon interactive, de les exécuter sur PostgreSQL™ et de voir les résultats de ces requêtes. Alternativement, les entrées peuvent êtres lues à partir d'un fichier. De plus, il fournit un certain nombre de méta-commandes et plusieurs fonctionnalités style shell pour faciliter l'écriture des scripts et automatiser un nombre varié de tâches.

Options

`-a`, `--echo-all`

Affiche toutes les lignes non vides en entrée sur la sortie standard lorsqu'elles sont lues. (Ceci ne s'applique pas aux lignes lues de façon interactive.) C'est équivalent à initialiser la variable `ECHO` à `all`.

`-A`, `--no-align`

Bascule dans le mode d'affichage non aligné. (Le mode d'affichage par défaut est aligné.)

`-c` *`commande`*, `--command` *`commande`*

Indique que psql doit exécuter une chaîne de commande, *`commande`*, puis s'arrêter. Cette option est utile dans les scripts shell. Les fichiers de démarrage (`psqlrc` et `~/.psqlrc`) sont ignorés avec cette option.

`commande` doit être soit une chaîne de commandes complètement analysable par le serveur (c'est-à-dire qui ne contient aucune des fonctionnalités spécifiques de psql), soit une seule commande antislash. Du coup, vous ne pouvez pas mixer les commandes SQL et les méta-commandes psql avec cette option. Pour réussir ceci, vous pouvez envoyer la chaîne dans un tube vers psql, par exemple : `echo "\x \\ SELECT * FROM foo;" | psql`. (`\\` est la méta-commande séparateur.)

Si la chaîne de commandes contient plusieurs commandes SQL, elles sont traitées dans une seule transaction sauf si des commandes **BEGIN**/**COMMIT** explicites sont inclues dans la chaîne pour la diviser en plusieurs transactions. Ceci est différent du comportement adopté quand la même chaîne est envoyée dans l'entrée standard de psql. De plus, seul le résultat de la dernière commande SQL est renvoyé.

À cause de ces comportements historiques, placer plus d'une commande dans la chaîne fournie à l'option `-c` a souvent des résultats inattendus. Il est préférable de fournir plusieurs commandes sur l'entrée standard de psql, soit en utilisant echo comme illustré ci-dessus, soit avec un document shell en ligne, par exemple :

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

`-d` *`nombase`*, `--dbname` *`nombase`*

Indique le nom de la base de données où se connecter. Ceci est équivalent à spécifier *`nombase`* comme premier argument de la ligne de commande qui n'est pas une option.

Si ce paramètre contient un signe =, il est traité comme une chaîne *conninfo*. Voir Section 31.1, « Fonctions de contrôle de connexion à la base de données » pour plus d'informations.

-e, --echo-queries

Copie toutes les commandes qui sont envoyées au serveur sur la sortie standard. Ceci est équivalent à initialiser la variable ECHO à queries.

-E, --echo-hidden

Affiche les requêtes réelles générées par **\d** et autres commandes antislash. Vous pouvez utiliser ceci pour étudier les opérations internes de psql. Ceci est équivalent à initialiser la variable ECHO_HIDDEN à on.

-f *nomfichier*, --file *nomfichier*

Utilise le fichier *nomfichier* comme source des commandes au lieu de lire les commandes de façon interactive. Une fois que le fichier est entièrement traité, psql se termine. Ceci est globalement équivalent à la commande interne **\i**.

Si *nomfichier* est un - (tiret), alors l'entrée standard est lue.

Utiliser cette option est légèrement différent d'écrire psql < *nomfichier*. En général, les deux feront ce que vous souhaitez mais utiliser -f active certaines fonctionnalités intéressantes comme les messages d'erreur avec les numéros de ligne. Il y a aussi une légère chance qu'utiliser cette option réduira la surcharge du lancement. D'un autre côté, la variante utilisant la redirection de l'entrée du shell doit (en théorie) pour ramener exactement le même affichage que celui que vous auriez eu en saisissant tout manuellement.

-F *séparateur*, --field-separator *séparateur*

Utilisez *séparateur* comme champ séparateur pour un affichage non aligné. Ceci est équivalent à **\pset fieldsep** ou **\f**.

-h *nomhôte*, --host *nomhôte*

Indique le nom d'hôte de la machine sur lequel le serveur est en cours d'exécution. Si la valeur commence avec un slash, elle est utilisée comme répertoire du socket de domaine Unix.

-H, --html

Active l'affichage en tableau HTML. Ceci est équivalent à \pset format html ou à la commande **\H**.

-l, --list

Liste toutes les bases de données disponibles puis quitte. Les autres option non relatives à la connexion sont ignorées. Ceci est similaire à la commande interne **\list**.

-L *nomfichier*, --log-file *nomfichier*

Écrit tous les résultats des requêtes dans le fichier *nomfichier* en plus de la destination habituelle.

-n, --no-readline

N'utilise pas Readline pour l'édition de lignes et n'utilise pas l'historique. Ceci est utile quand on veut désactiver la gestion de la tabulation pour l'action copie/colle.

-o *nomfichier*, --output *nomfichier*

Dirige tous les affichages de requêtes dans le fichier *nomfichier*. Ceci est équivalent à la commande **\o**.

-p *port*, --port *port*

Indique le port TCP ou l'extension du fichier socket de domaine local Unix sur lequel le serveur attend les connexions. Par défaut, il s'agit de la valeur de la variable d'environnement PGPORT ou, si elle n'est pas initialisée, le port spécifié au moment de la compilation, habituellement 5432.

-P *affectation*, --pset *affectation*

Vous permet de spécifier les options d'affichage dans le style de **\pset** sur la ligne de commande. Notez que, ici, vous devez séparer nom et valeur avec un signe égal au lieu d'un espace. Du coup,

pour initialiser le format d'affichage en LaTeX, vous devez écrire `-P format=latex`.

-q, --quiet

Indique que psql doit travailler silencieusement. Par défaut, il affiche des messages de bienvenue et des informations diverses. Si cette option est utilisée, rien de ceci n'est affiché. C'est utile avec l'option `-c`. C'est équivalent à configurer la variable `QUIET` à `on`.

-R *séparateur*, --record-separator *séparateur*

Utilisez *séparateur* comme séparateur d'enregistrement pour un affichage non aligné. Ceci est équivalent à la commande **\pset recordsep**.

-s, --single-step

S'exécute en mode étape par étape. Ceci signifie qu'une intervention de l'utilisateur est nécessaire avant l'envoi de chaque commande au serveur, avec une option pour annuler l'exécution. Utilisez cette option pour déboguer des scripts.

-S, --single-line

S'exécute en mode simple ligne où un retour à la ligne termine une commande SQL, de la même façon qu'un point-virgule.

Note

Ce mode est fourni pour ceux qui insistent pour l'avoir, mais vous n'êtes pas nécessairement encouragé à l'utiliser. En particulier, si vous mixez SQL et méta-commandes sur une ligne, l'ordre d'exécution n'est pas toujours clair pour l'utilisateur non expérimenté.

-t, --tuples-only

Désactive l'affichage des noms de colonnes et le pied de page contenant le nombre de résultats, etc. Ceci est équivalent à la méta-commande **\t**.

-T *options_table*, --table-attr *options_table*

Permet d'indiquer les options à placer à l'intérieur d'une balise `table` en HTML. Voir **\pset** pour plus de détails.

-U *nomutilisateur*, --username *nomutilisateur*

Se connecte à la base de données en tant que l'utilisateur *nomutilisateur* au lieu de celui par défaut. (Vous devez aussi avoir le droit de le faire, bien sûr.)

-v *affectation*, --set *affectation*, --variable *affectation*

Réalise une affectation de variable comme la commande interne **\set**. Notez que vous devez séparer nom et valeur par un signe égal sur la ligne de commande. Pour désinitialiser une variable, enlevez le signe d'égalité. Pour simplement initialiser une variable sans valeur, utilisez le signe égal sans passer de valeur. Ces affectations sont réalisées lors de la toute première étape du lancement, du coup les variables réservées à des buts internes peuvent être écrasées plus tard.

-V, --version

Affiche la version de psql et quitte.

-w, --no-password

Ne demande jamais un mot de passe. Si le serveur en réclame un pour l'authentification et qu'un mot de passe n'est pas disponible d'une autre façon (par exemple avec le fichier `.pgpass`), la tentative de connexion échouera. Cette option peut être utile pour les scripts où aucun utilisateur n'est présent pour saisir un mot de passe.

Notez que cette option restera positionnée pour l'ensemble de la session, et qu'elle affecte aussi l'utilisation de la méta-commande **\connect** en plus de la tentative de connexion initiale.

-W, --password

Force psql à demander un mot de passe avant de se connecter à une base de données.

Cette option n'est jamais obligatoire car psql demandera automatiquement un mot de passe si le serveur exige une authentification par mot de passe. Néanmoins, psql perdra une tentative de connexion pour trouver que le serveur veut un mot de passe. Dans certains cas, il est préférable d'ajouter l'option `-W` pour éviter la tentative de connexion.

Notez que cette option sera conservée pour la session entière, et que du coup elle affecte l'utilisation de la méta-commande **\connect** ainsi que la tentative de connexion initiale.

-x, --expanded

Active le mode de formatage de table étendu. Ceci est équivalent à la commande **\x**.

-X,, --no-psqlrc

Ne lit pas le fichier de démarrage (ni le fichier système `psqlrc` ni celui de l'utilisateur `~/.psqlrc`).

-1, --single-transaction

Quand psql exécute un script avec l'option `-f`, ajouter cette option englobe le script avec les instructions **BEGIN/COMMIT** pour tout faire dans une seule transaction. Ceci nous assure que soit toutes les commandes se terminent avec succès, soit aucune modification n'est enregistrée.

Si le script utilise lui-même **BEGIN**, **COMMIT** ou **ROLLBACK**, cette option n'aura pas les effets désirés. De plus, si le script contient toute commande qui ne peut pas être exécutée à l'intérieur d'un bloc de transaction, indiquer cette option provoquera l'échec de cette commande (et du coup de la transaction entière).

-?, --help

Affiche de l'aide sur les arguments en ligne de commande de psql et quitte.

Code de sortie

psql renvoie 0 au shell s'il se termine normalement, 1 s'il y a eu une erreur fatale de son fait (pas assez de mémoire, fichier introuvable), 2 si la connexion au serveur s'est interrompue ou a été annulée, 3 si une erreur est survenue dans un script et si la variable `ON_ERROR_STOP` a été initialisée.

Usage

Se connecter à une base de données

psql est une application client PostgreSQL™ standard. Pour se connecter à une base de données, vous devez connaître le nom de votre base de données cible, le nom de l'hôte et le numéro de port du serveur ainsi que le nom de l'utilisateur que vous souhaitez connecter. psql peut connaître ces paramètres à partir d'options en ligne de commande, respectivement `-d`, `-h`, `-p` et `-U`. Si un argument autre qu'une option est rencontré, il est interprété comme le nom de la base de données (ou le nom de l'utilisateur si le nom de la base de données est déjà donné). Toutes les options ne sont pas requises, des valeurs par défaut sont applicables. Si vous omettez le nom de l'hôte, psql se connecte via un socket de domaine Unix à un serveur sur l'hôte local ou via TCP/IP sur `localhost` pour les machines qui n'ont pas sockets de domaine Unix. Le numéro de port par défaut est déterminé au moment de la compilation. Comme le serveur de bases de données utilise la même valeur par défaut, vous n'aurez pas besoin de spécifier le port dans la plupart des cas. Le nom de l'utilisateur par défaut est votre nom d'utilisateur du système d'exploitation, de même pour le nom de la base de données par défaut. Notez que vous ne pouvez pas simplement vous connecter à n'importe quelle base de données avec n'importe quel nom d'utilisateur. Votre administrateur de bases de données doit vous avoir informé de vos droits d'accès.

Quand les valeurs par défaut ne sont pas correctes, vous pouvez vous simplifier la vie en configurant les variables d'environnement `PGDATABASE`, `PGHOST`, `PGPORT` et/ou `PGUSER` avec les valeurs appropriées (pour les variables d'environnement supplémentaires, voir Section 31.13, « Variables d'environnement »). Il est aussi intéressant d'avoir un fichier `~/.pgpass` pour éviter d'avoir régulièrement à saisir des mots de passe. Voir Section 31.14, « Fichier de mots de passe » pour plus d'informations.

Une autre façon d'indiquer les paramètres de connexion est dans une chaîne *`conninfo`* qui est utilisée à la place du nom d'une base de données. Ce mécanisme vous donne un grand contrôle sur la connexion. Par exemple :

```
$ psql "service=monservice sslmode=require"
```

De cette façon, vous pouvez aussi utiliser LDAP pour la recherche de paramètres de connexion, comme décrit dans Section 31.16, « Recherches LDAP des paramètres de connexion ». Voir Section 31.1, « Fonctions de contrôle de connexion à la base de données » pour plus d'informations sur toutes les options de connexion disponibles.

Si la connexion ne peut pas se faire, quelle qu'en soit la raison (c'est-à-dire droits non suffisants, serveur arrêté sur l'hôte cible, etc.), psql renvoie une erreur et s'arrête.

Saisir des commandes SQL

Dans le cas normal, psql fournit une invite avec le nom de la base de données sur laquelle psql est connecté suivi par la chaîne =>. Par exemple

```
$ psql basetest
psql (9.0.23)
Type "help" for help.

basetest=>
```

À l'invite l'utilisateur peut saisir des commandes SQL. Ordinairement, les lignes en entrée sont envoyées vers le serveur quand un point-virgule de fin de commande est saisi. Une fin de ligne ne termine pas une commande. Du coup, les commandes peuvent être saisies sur plusieurs lignes pour plus de clarté. Si la commande est envoyée et exécutée sans erreur, les résultats de la commande sont affichés sur l'écran.

À chaque fois qu'une commande est exécutée, psql vérifie aussi les événements de notification générés par LISTEN(7) et NOTIFY(7).

Meta-commandes

Tout ce que vous saisissez dans psql qui commence par un antislash non échappé est une méta-commande psql qui est traitée par psql lui-même. Ces commandes aident à rendre psql plus utile pour l'administration ou pour l'écriture de scripts. Les méta-commandes sont plus souvent appelées les commandes slash ou antislash.

Le format d'une commande psql est l'antislash suivi immédiatement d'un verbe de commande et de ses arguments. Les arguments sont séparés du verbe de la commande et les uns des autres par un nombre illimité d'espaces blancs.

Pour inclure des espaces blancs dans un argument, vous devez l'envelopper dans des guillemets simples. Pour inclure un guillemet simple dans un argument, utilisez deux guillemets simples. Tout ce qui est contenu entre des guillemets simples est de plus sujet à des substitutions style C pour `\n` (nouvelle ligne), `\t` (tabulation), `\`*`chiffres`* (octal) et `\x`*`chiffres`* (hexadécimal).

Si un argument sans guillemets commence avec un caractère `:`, il est pris pour une variable psql et la valeur de la variable est utilisée à la place de l'argument.

Les arguments placés entre guillemets arrières (`) sont pris comme une ligne de commande passée au shell. L'affichage de la commande (sans l'éventuel saut de ligne à la fin) est pris comme valeur de l'argument. Cela s'applique aussi aux séquences d'échappement ci-dessus.

Quelques commandes prennent un identifiant SQL (comme un nom de table) en argument. Ces arguments suivent les règles de la syntaxe SQL : les lettres sans guillemets sont forcées en minuscule alors que les guillemets doubles (") protègent les lettres de la conversion de casse et autorisent l'incorporation d'espaces blancs dans l'identifiant. À l'intérieur des guillemets doubles, les guillemets doubles en paire se réduisent à un seul guillemet double dans le nom résultant. Par exemple, `FOO"BAR"BAZ` est interprété comme `fooBARbaz` et `"Un nom ""bizarre"` devient `Un nom "bizarre`.

L'analyse des arguments se termine quand d'autres antislash non entre guillemets surviennent. Ceci est pris pour le début d'une nouvelle méta-commande. La séquence spéciale `\\` (deux antislashes) marque la fin des arguments et continue l'analyse des commandes SQL, si elles existent. De cette façon, les commandes SQL et psql peuvent être mixées librement sur une ligne. Mais dans tous les cas, les arguments d'une meta-commande ne peuvent pas continuer après la fin de la ligne.

Les meta-commandes suivantes sont définies :

`\a`

Si le format actuel d'affichage d'une table est non aligné, il est basculé à aligné. S'il n'est pas non aligné, il devient non aligné. Cette commande est conservée pour des raisons de compatibilité. Voir **\pset** pour une solution plus générale.

\cd [*répertoire*]

Modifie le répertoire courant par *répertoire*. Sans argument, le répertoire personnel de l'utilisateur devient le répertoire courant.

Astuce

Pour afficher votre répertoire courant, utilisez `\! pwd`.

\C [*titre*]

Initialise ou supprime le titre des tables affichées en résultat d'une requête. Cette commande est équivalente à `\pset title` *titre*. (Le nom de cette commande provient de « caption », car elle avait précédemment pour seul but d'initialiser l'en-tête dans une table HTML.)

\connect (or \c) [*nom_base* [*nom_utilisateur*] [*hôte*] [*port*]]

Établie une nouvelle connexion à un serveur PostgreSQL™. Les paramètres de connexion utilisés peuvent être spécifiés en utilisant soit la syntaxe par position soit les chaînes de connexion `conninfo` telles qu'elles sont détaillées dans Section 31.1, « Fonctions de contrôle de connexion à la base de données ».

Lors de l'utilisation de paramètres de position, si un paramètre parmi *dbname*, *username*, *host* ou *port* sont omis ou spécifiés sous la forme d'un tiret (`-`), la valeur utilisée pour ce paramètre est celui de la précédente connexion ; s'il n'y a pas de connexion précédente, la valeur par défaut de la libpq est utilisé. Lors de l'utilisation de chaînes `conninfo`, les valeurs de la précédente connexion ne sont pas utilisées pour la nouvelle connexion.

Si la nouvelle connexion est réussie, la connexion précédente est fermée. Si la tentative de connexion échoue (mauvais nom d'utilisateur, accès refusé, etc.), la connexion précédente est conservée si psql est en mode interactif. Lors de l'exécution d'un script non interactif, le traitement s'arrêtera immédiatement avec une erreur. Cette distinction a été choisie pour deux raisons : aider l'utilisateur face aux fautes de frappe et en tant que mesure de précaution pour qu'un script n'agisse pas par erreur sur la mauvaise base.

Exemples :

```
=> \c mydb myuser host.dom 6432
=> \c service=foo
=> \c "host=localhost port=5432 dbname=mydb connect_timeout=10 sslmode=dis
```

\copy { *table* [(*liste_colonnes*)] | (*requête*) } { from | to } { *nomfichier* | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] '*caractère*'] [null [as] '*chaîne*'] [csv [header] [quote [as] '*caractère*'] [escape [as] '*caractère*'] [force quote *liste_colonnes*] [force not null *liste_colonnes*]]

Réalise une opération de copy côté client. C'est une opération qui exécute une commande SQL, COPY(7), mais au lieu que le serveur lise ou écrive le fichier spécifié, psql lit ou écrit le fichier en faisant le routage des données entre le serveur et le système de fichiers local. Ceci signifie que l'accès et les droits du fichier sont ceux de l'utilisateur local, pas celui du serveur, et qu'aucun droit de superutilisateur n'est requis.

La syntaxe de la commande est similaire à celle de la commande COPY(7) SQL. Notez que, à cause de cela, des règles spéciales d'analyse s'appliquent à la commande **\copy**. En particulier, les règles de substitution de variable et d'échappement antislash ne s'appliquent pas.

`\copy ... from stdin | to stdout` lit/écrit basé sur l'entrée standard de la commande ou sa sortie standard respectivement. Toutes les lignes sont lues à partir de la même source qui a lancé la commande, en continuant jusqu'à ce que `\.` soit lu ou que le flux parvienne à EOF. La sortie est envoyée au même endroit que la sortie de la commande. Pour lire/écrire à partir de l'entrée et de la sortie standard de psql, utilisez `pstdin` ou `pstdout`. Cette option est utile pour peupler des tables en ligne à l'intérieur d'un fichier script SQL.

Astuce

Cette opération n'est pas aussi efficace que la commande **COPY** en SQL parce que toutes les données doivent passer au travers de la connexion client/serveur. Pour les grosses masses de données, la commande SQL est préférable.

\copyright

Affiche le copyright et les termes de distribution de PostgreSQL™.

\d[S+] [*motif*]

Pour chaque relation (table, vue, index ou séquence) correspondant au *motif*, affiche toutes les colonnes, leur types, le tablespace (s'il ne s'agit pas du tablespace par défaut) et tout attribut spécial tel que `NOT NULL` ou les valeurs par défaut. Les index, contraintes, règles et déclencheurs associés sont aussi affichés, ainsi que la définition de la vue si la relation est une vue. (Ce qui « Correspond au motif » est défini ci-dessous.)

Le forme de la commande `\d+` est identique, sauf que des informations plus complètes sont affichées : tout commentaire associé avec les colonnes de la table est affiché, ainsi que la présence d'OID dans la table.

Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

Note

Si **\d** est utilisé sans argument *motif*, c'est équivalent, en plus commode, à **\dtvs** qui affiche une liste de toutes les tables, vues et séquences.

\da[S] [*motif*]

Liste toutes les fonctions d'agrégat disponibles, avec lee type de retour et les types de données sur lesquels elles opèrent. Si *motif* est spécifié, seuls les agrégats dont les noms commencent par le motif sont affichés. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

\db[+] [*motif*]

Liste tous les tablespaces disponibles. Si *motif* est spécifié, seuls les tablespaces dont le nom correspond au motif sont affichés. Si + est ajouté à la fin de la commande, chaque objet est listé avec les droits associés.

\dc[S] [*motif*]

Liste les conversions entre les encodages de jeux de caractères. Si *motif* est spécifié, seules les conversions dont le nom correspond au motif sont listées. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

\dC [*motif*]

Liste les conversions de types. Si *motif* est indiqué, seules sont affichées les conversions dont le type source ou cible correspond au motif.

\dd[S] [*motif*]

Affiche les descriptions des objets correspondant au *motif* ou de tous les objets si aucun argument n'est donné. Mais dans tous les cas, seuls les objets qui ont une description sont listés. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes. Le terme « objet » couvre les agrégats, les fonctions, les opérateurs, les types, les relations (tables, vues, index, séquences, objets larges), les règles et les déclencheurs. Par exemple, :

```
=> \dd version
                        Object descriptions
   Schema   |  Name   |  Object  |        Description
```

```
------------+---------+----------+--------------------------
 pg_catalog | version | function | PostgreSQL version string
(1 row)
```

Les descriptions des objets peuvent être ajoutées avec la commande SQL COMMENT(7).

\ddp [*motif*]

Liste les paramètres par défaut pour les privilèges d'accès. Une entrée est affichée pour chaque rôle (et schéma, si c'est approprié) pour lequel les paramètres par défaut des privilèges ont été modifiés par rapport aux paramètres par défaut intégrés. Si *motif* est spécifié, seules les entrées dont le nom de rôle ou le nom de schéma correspond au motif sont listées.

La commande ALTER DEFAULT PRIVILEGES(7) sert à positionner les privilèges d'accès par défaut. Le sens de l'affichage des privilèges est expliqué à la page de GRANT(7).

\dD[S] [*motif*]

Liste les domaines. Si *motif* est spécifié, seuls les domaines dont le nom correspond au motif sont affichés. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

\des[+] [*motif*]

Liste les serveurs distants (mnémonique : « external servers »). Si *motif* est spécifié, seuls les serveurs dont le nom correspond au motif sont affichés. Si la forme `\des+` est utilisée, une description complète de chaque serveur est affichée, incluant la liste de contrôle d'accès du serveur (ACL), type, version et options.

\deu[+] [*motif*]

Liste les correspondances d'utilisateurs (mnémonique : « external users »). Si *motif* est spécifié, seules les correspondances dont le nom correspond au motif sont affichées. Si la forme `\deu+` est utilisée, des informations supplémentaires sur chaque correspondance d'utilisateur sont affichées.

Attention

`\deu+` risque aussi d'afficher le nom et le mot de passe de l'utilisateur distant, il est donc important de faire attention à ne pas les divulguer.

\dew[+] [*motif*]

Liste les wrappers de données distants (mnémonique : « external wrappers »). Si *motif* est spécifié, seuls les wrappers dont le nom correspond au motif sont affichés. Si la forme `\dew+` est utilisée, les ACL et options du wrapper sont aussi affichées.

\df[antwS+] [*pattern*]

Liste les fonctions, ainsi que leurs arguments, types de retour, et types de fonctions, qui sont classés comme « agg » (agrégat), « normal », « trigger », or « window ». Afin de n'afficher que les fonctions d'un type spécifié, ajoutez les lettres correspondantes, respectivement `a`, `n`, `t`, or `w` à la commande. Si *motif* est spécifié, seules les fonctions dont le nom correspond au motif sont affichées. Si la forme `\df+` est utilisée, des informations supplémentaires sur chaque fonction, dont la volatibilité, le langage, le code source et la description, sont proposées. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

Astuce

Pour rechercher des fonctions prenant des arguments ou des valeurs de retour d'un type spécifique, utilisez les capacités de recherche du paginateur pour parcourir la sortie de `\df`.

\dF[+] [*motif*]

Liste les configurations de la recherche plein texte. Si *motif* est spécifié, seules les configurations dont le nom correspond au motif seront affichées. Si la forme `\dF+` est utilisée, une description complète de chaque configuration est affichée, ceci incluant l'analyseur de recherche plein texte et la liste de dictionnaire pour chaque type de jeton de l'analyseur.

\dFd[+] [*motif*]

Liste les dictionnaires de la recherche plein texte. Si *motif* est spécifié, seuls les dictionnaires dont le nom correspond au motif seront affichés. Si la forme `\dFd+` est utilisée, des informations supplémentaires sont affichées pour chaque dictionnaire, ceci incluant le motif de recherche plein texte et les valeurs des options.

\dFp[+] [*motif*]

Liste les analyseurs de la recherche plein texte. Si *motif* est spécifié, seuls les analyseurs dont le nom correspond au motif seront affichés. Si la forme `\dFp+` est utilisée, une description complète de chaque analyseur est affichée, ceci incluant les fonctions sous-jacentes et les types de jeton reconnu.

\dFt[+] [*motif*]

Liste les motifs de la recherche plein texte. Si *motif* est spécifié, seuls les motifs dont le nom correspond au motif seront affichés. Si la forme `\dFt+` est utilisée, des informations supplémentaires sont affichées pour chaque motif, ceci incluant les noms des fonctions sous-jacentes.

\dg[+] [*motif*]

Liste les rôles des bases de données. Si *motif* est spécifié, seuls les rôles dont le nom correspond au motif sont listés. (Cette commande est maintenant réellement identique à `\du`.) Si la forme `\dg+` est utilisée, des informations supplémentaires sont affichées pour chaque rôle, par exemple le commentaire.

\di[S+] [*motif*], \ds[S+] [*motif*], \dt[S+] [*motif*], \dv[S+] [*motif*]

Dans ce groupe de commandes, les lettres `i`, `s`, `t` et `v` correspondent respectivement à index, séquence, table et vue. Vous pouvez indiquer n'importe quelle combinaison de ces lettres, dans n'importe quel ordre, pour obtenir la liste de tous les objets de ces types. Par exemple, `\dit` liste les index et tables. Si + est ajouté à la fin de la commande, chaque objet est listé avec sa taille physique sur disque et sa description associée s'il y en a une. Si *motif* est spécifié, seuls les objets dont les noms correspondent au motif sont listés. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

\dl

Ceci est un alias pour **\lo_list**, qui affiche une liste des objets larges.

\dn[+] [*motif*]

Liste les schémas. Si *motif* est spécifié, seuls les schémas dont le nom correspond au motif sont listés. Tout schéma temporaire non local est supprimé. Si + est ajouté à la fin de la commande, chaque objet est listé avec ses droits et description associés.

\do[S] [*motif*]

Liste les opérateurs avec leur opérande et type en retour. Si *motif* est spécifié, seuls les opérateurs dont le nom correspond au motif sont listés. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur `S` pour afficher les objets systèmes.

\dp [*motif*]

Liste les tables, vues et séquences avec leur droits d'accès associés. Si *motif* est spécifié, seules les tables, vues et séquences dont le nom correspond au motif sont listées.

Les commandes GRANT(7) et REVOKE(7) sont utilisées pour configurer les droits d'accès. Les explications sur le sens de l'affichage des privilèges sont sous GRANT(7).

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

Liste les paramètres de configuration définis. Ces paramètres peuvent être spécifiques à un rôle, spécifiques à une base, ou les deux. *role-pattern* et *database-pattern* servent à choisir sur quels rôles spécifiques ou quelles bases de données - respectivement - les paramètres sont listés. Si ces

options sont omises, ou si on spécifie *, tous les paramètres sont listés, y compris ceux qui ne sont pas spécifiques à un rôle ou à une base, respectivement.

Les commande ALTER ROLE(7) et ALTER DATABASE(7) servent à définir les paramètres de configuration par rôle et par base de données.

\dT[S+] [*motif*]

Liste les types de données. Si *motif* est spécifié, seuls les types dont le nom correspond au motif sont affichés. Si + est ajouté à la fin de la commande, chaque type est listé avec son nom interne et sa taille, ainsi que ses valeurs autorisées si c'est un type enum. Par défaut, seuls les objets créés par les utilisateurs sont affichés ; fournissez un motif ou le modificateur S pour afficher les objets systèmes.

\du [*motif*]

Liste les rôles de la base de données. Si *motif* est spécifié, seuls les rôles dont le nom correspond au motif sont affichés. Si la forme \du+ est utilisée, des informations supplémentaires sont affichées pour chaque rôle, par exemple le commentaire.

\edit (ou \e) [*nomfichier*]

Si *nomfichier* est spécifié, le fichier est édité ; en quittant l'éditeur, son contenu est recopié dans le tampon de requête. Si aucun argument n'est fourni, le tampon de requête actuel est copié dans un fichier temporaire qui est ensuite édité de la même façon.

Le nouveau tampon de requête est ensuite ré-analysé suivant les règles habituelles de psql, où le tampon complet est traité comme une seule ligne. (Du coup, vous ne pouvez pas faire de scripts de cette façon. Utilisez **\i** pour cela.) Ceci signifie aussi que si la requête se termine avec (ou plutôt contient) un point-virgule, elle est immédiatement exécutée. Dans les autres cas, elle attend simplement dans le tampon de requête.

Astuce

psql recherche les variables d'environnement PSQL_EDITOR, EDITOR et VISUAL (dans cet ordre) pour connaître l'éditeur à utiliser. Si aucun n'est initialisé, vi est utilisé sur les systèmes Unix, notepad.exe sur les systèmes Windows.

\ef [*description_fonction*]

Cette commande récupère et édite la définition de la fonction désignée au moyen d'une commande **CREATE OR REPLACE FUNCTION**. L'édition est faite de la même façon que pour \e. Après que l'éditeur se soit fermé, la commande mise à jour attend dans le tampon de requête ; tapez ; ou \g pour l'envoyer, ou \r pour l'annuler.

La fonction cible peut être spécifiée par son nom seul, ou par son nom et ses arguments, par exemple foo(integer, text). Les types d'arguments doivent être fournis s'il y a plus d'une fonction du même nom.

Si aucune fonction n'est spécifiée, un modèle d'ordre **CREATE FUNCTION** vierge est affiché pour édition.

\echo *texte* [...]

Affiche les arguments sur la sortie standard séparés par un espace et suivi par une nouvelle ligne. Ceci peut être utile pour intégrer des informations sur la sortie des scripts. Par exemple :

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

Si le premier argument est -n sans guillemets, alors la fin de ligne n'est pas écrite.

Astuce

Si vous utilisez la commande **\o** pour rediriger la sortie de la requête, vous pourriez souhaiter utiliser **\qecho** au lieu de cette commande.

\encoding [*codage*]

Initialise l'encodage du jeu de caractères du client. Sans argument, cette commande affiche l'encodage actuel.

\f [*chaîne*]

Initialise le champ séparateur pour la sortie de requête non alignée. La valeur par défaut est la barre verticale (|). Voir aussi **\pset** comme moyen générique de configuration des options d'affichage.

\g [*nomfichier*], \g [|*commande*]

Envoie le tampon de requête en entrée vers le serveur et stocke en option la sortie de la requête dans *nomfichier* ou envoie dans un tube la sortie vers un shell exécutant *commande*. Un simple \g est virtuellement équivalent à un point-virgule. Un \g avec argument est une alternative en « un coup » à la commande **\o**.

\help (ou \h) [*commande*]

Donne la syntaxe sur la commande SQL spécifiée. Si *commande* n'est pas spécifiée, alors psql liste toutes les commandes pour lesquelles une aide en ligne est disponible. Si *commande* est un astérisque (*), alors l'aide en ligne de toutes les commandes SQL est affichée.

Note

Pour simplifier la saisie, les commandes qui consistent en plusieurs mots n'ont pas besoin d'être entre guillemets. Du coup, il est correct de saisir **\help alter table**.

\H ou \html

Active le format d'affichage HTML des requêtes. Si le format HTML est déjà activé, il est basculé au format d'affichage défaut (texte aligné). Cette commande est pour la compatibilité mais voir **\pset** pour configurer les autres options d'affichage.

\i ou \include *nomfichier*

Lit l'entrée à partir du fichier *nomfichier* et l'exécute comme si elle avait été saisie sur le clavier.

Note

Si vous voulez voir les lignes sur l'écran au moment de leur lecture, vous devez initialiser la variable ECHO à all.

\l (ou \list), \l+ (ou \list+)

Liste les noms, propriétaires, encodage de jeux de caractères et droits d'accès de toutes les bases du serveur. Si + est ajouté à la fin de la commande, la taille des bases, les tablespaces par défaut et les descriptions sont aussi affichées. (Les tailles ne sont disponibles que pour les bases auxquelles l'utilisateur courant a le droit de se connecter.)

\lo_export *loid nomfichier*

Lit l'objet large d'OID *loid* à partir de la base de données et l'écrit dans *nomfichier*. Notez que ceci est subtilement différent de la fonction serveur lo_export, qui agit avec les droits de l'utilisateur avec lequel est exécuté le serveur de base de données et sur le système de fichiers du serveur.

Astuce

Utilisez **\lo_list** pour trouver l'OID de l'objet large.

\lo_import *nomfichier* [*commentaire*]

Stocke le fichier dans un objet large PostgreSQL™. En option, il associe le commentaire donné avec l'objet. Exemple :

```
foo=> \lo_import '/home/peter/pictures/photo.xcf' 'une
photo de moi'
lo_import 152801
```

La réponse indique que l'objet large a reçu l'ID 152801, qui peut être utilisé pour accéder de nouveau à l'objet créé. Pour une meilleure lisibilité, il est recommandé de toujours associer un commentaire compréhensible par un humain avec chaque objet. Les OID et les commentaires sont visibles avec la commande **\lo_list**.

Notez que cette commande est subtilement différente de la fonction serveur `lo_import` car elle agit en tant qu'utilisateur local sur le système de fichier local plutôt qu'en tant qu'utilisateur du serveur et de son système de fichiers.

\lo_list

Affiche une liste de tous les objets larges PostgreSQL™ actuellement stockés dans la base de données, avec tous les commentaires fournis par eux.

\lo_unlink *loid*

Supprime l'objet large d'OID *loid* de la base de données.

Astuce

Utilisez **\lo_list** pour trouver l'OID d'un objet large.

\o or \out [*nomfichier*], \o or \out [|*commande*]

S'arrange pour sauvegarder les résultats des prochaines requêtes dans le fichier *nomfichier* ou d'envoyer les résultats à la commande shell *commande*. Si aucun argument n'est fourni, le résultat de la requête va sur la sortie standard.

Les « résultats de requête » incluent toutes les tables, réponses de commande et messages d'avertissement obtenus du serveur de bases de données, ainsi que la sortie de différentes commandes antislash qui envoient des requêtes à la base de données (comme **\d**), mais sans message d'erreur.

Astuce

Pour intégrer du texte entre les résultats de requête, utilisez **\qecho**.

\p ou \print

Affiche le tampon de requête actuel sur la sortie standard.

\password [*nom_utilisateur*]

Modifie le mot de passe de l'utilisateur indiqué (par défaut, l'utilisateur en cours). Cette commande demande le nouveau mot de passe, le chiffre et l'envoie au serveur avec la commande **ALTER ROLE**. Ceci vous assure que le nouveau mot de passe n'apparaît pas en clair dans l'historique de la commande, les traces du serveur ou encore ailleurs.

\prompt [*texte*] *nom*

Demande à l'utilisateur la valeur pour la variable *nom*. Un affichage optionnel, *texte*, peut être proposé. (Pour des invites à plusieurs mots, utilisez les guillemets simples.)

Par défaut, `\prompt` utilise le terminal pour les entrées et sorties. Néanmoins, si la bascule `-f` est utilisée, `\prompt` utilise l'entrée et la sortie standard.

\pset *option* [*valeur*]

Cette commande initialise les options affectant l'affichage des tables résultat de la requête. *option* décrit l'option à initialiser. La sémantique de *valeur* varie en fonction de l'option sélectionnée. Pour certaines options, omettre *valeur* a pour conséquence de basculer ou désactiver l'option, tel que cela

est décrit pour chaque option. Si aucun comportement de ce type n'est mentionné, alors omettre *valeur* occasionne simplement l'affichage de la configuration actuelle.

Les options ajustables d'affichage sont :

format

Initialise le format d'affichage parmi unaligned, aligned, wrapped, html, latex ou troff-ms. Les abréviations uniques sont autorisées. (ce qui signifie qu'une lettre est suffisante.)

Le format unaligned écrit toutes les colonnes d'un enregistrement sur une seule ligne, séparées par le séparateur de champ courant. Ceci est utile pour créeer des sorties qui doivent être lues par d'autres programmes (au format séparé par des caractère tabulation ou par des virgules, par exemple).

Le format aligned est le format de sortie texte standard, lisible par les humains, joliement formaté ; c'est le format par défaut.

Le format wrapped est comme aligned, sauf qu'il retourne à la ligne dans les données de grande taille afin que la sortie tienne dans la largeur de colonne cible. La largeur cible est déterminée comme cela est décrit à l'option columns. Notez que psql n'essaie pas de revenir à la ligne dans les titres de colonnes. Par conséquent, si la largeur totale nécessaire pour le titre de colonne est plus grande que la largeur cible, le format wrapped se comporte de la même manière que aligned.

Les formats html, latex, and troff-ms produisent des tables destinées à être inclues dans des documents utilisant le langage de marques respectif. Ce ne sont pas des documents complets ! (Ce n'est pas dramatique en HTML mais en LaTeX vous devez avoir une structure de document complet.)

border

Le *valeur* doit être un nombre. En général, plus grand est ce nombre, plus les tables ont de bordure et de ligne mais ceci dépend du format. Dans le mode HTML, ceci sera traduit directement avec l'attribut border=.... Avec les autres, seules les valeurs 0 (sans bordure), 1 (lignes internes de division) et 2 (forme de table) ont un sens.

columns

Positionne la largeur pour le format wrapped , ainsi que la largeur à partir de laquelle la sortie est suffisemment longue pour nécessiter le pager. Si l'option est positionnée à zéro (la valeur par défaut), la largeur de la colonne est contrôlée soit par la variable d'environnement COLUMNS, soit par la largeur d'écran détectée si COLUMNS n'est pas positionnée. De plus, si columns vaut zero, alors le format wrapped affecte seulement la sortie écran. Si columns ne vaut pas zéro, alors les sorties fichier et tubes (pipe) font l'objet de retours à la ligne à cette largeur également.

linestyle

Positionne le style des lignes de bordure sur ascii, old-ascii unicode. Les abréviations uniques sont autorisées. (Cela signifie qu'une lettre suffit.) La valeur par défaut est ascii. Cette option affecte seulement les formats de sortie aligned et wrapped.

Le style ascii utilise les caractères basiques ASCII . Les retours à la ligne dans les données sont représentés par un symbole + dans la marge de droite. Si le format wrapped est sélectionné, un retour chariot est ajouté à l'affichage pour les valeurs dont la taille à l'affichage est trop importante pour tenir dans une cellule de la colonne associée. Un point (.) est affiché dans la marge droite de la ligne avant le retour chariot et un autre point est affiché dans la marge gauche de la ligne suivante.

Le style old-ascii utilise des caractères basiques ASCII, utilisant le style de formatage utilisé dans PostgreSQL™ 8.4 and et les versions plus anciennes. Les retours à la ligne dans les données sont représentés par un symbole : à la place du séparateur de colonnes placé à gauche. Quand les données sont réparties sur plusieurs lignes sans qu'il y ait de caractère de retour à la ligne dans les données, un symbole ; est utilisé à la place du séparateur de colonne de gauche.

Le style `unicode` utilise les caractères Unicode de dessin de boîte. Les retours à la ligne dans les données sont représentés par un symbole de retour à la ligne dans la marge de droite. Lorsque les données sont réparties sur plusieurs lignes, sans qu'il y ait de caractère de retour à la ligne dans les données, le symbole ellipse est affiché dans la marge de droite de la première ligne, et également dans la marge de gauche de la ligne suivante.

Quand le paramètre `border` vaut plus que zéro, cette option détermine également les caractères utilisés pour dessiner les lignes de bordure. Les simples caractères ASCII fonctionnent partout, mais les caractères Unicode sont plus jolis sur les affichages qui les reconnaissent.

`expanded` (ou `x`)

Si le paramètre *`valeur`* est précisé, il peut valoir soit `on` soit `off` qui activera ou désactivera le mode étendu. Si *`valeur`* est omis, la commande bascule entre le mode normal et le mode étendu. Quand le mode étendu est activé, les résultats d'une requête sont affichés sur deux colonnes, avec le nom de la colonne dans la partie gauche et les données dans la partie droite. Ce mode est utile si les données ne tiennent pas sur l'écran dans le mode normal, « horizontal ».

`null`

Positionne la chaîne de caractères à afficher à la place d'une valeur null. Par défaut, rien n'est affiché, ce qui peut facilement être confondu avec une chaîne de caractères vide. Par exemple, vous pouvez préférer afficher `\pset null '(null)'`.

`fieldsep`

Indique le séparateur de champ à utiliser dans le mode d'affichage non aligné. De cette façon, vous pouvez créer, par exemple, une sortie séparée par des tabulations ou des virgules, que d'autres programmes pourraient préférer. Pour configurer une tabulation comme champ séparateur, saisissez `\pset fieldsep '\t'`. Le séparateur de champ par défaut est `'|'` (une barre verticale).

`footer`

Si le paramètre *`valeur`* est précisé, il doit valoir soit `on`, soit `off`, ce qui a pour effet d'activer ou de désactiver l'affichage du pied de table (le compte : `(`*`n`*` rows)`). Si le paramètre *`valeur`* est omis, la commande bascule entre l'affichage du pied de table ou sa désactivation.

`numericlocale`

Si *`valeur`* est précisée, elle doit valoir soit `on`, soit `off` afin d'activer ou désactiver l'affichage d'un caractère dépendant de la locale pour séparer des groupes de chiffres à gauche du séparateur décimal. Si *`valeur`* est omise, la commande bascule entre la sortie numérique classique et celle spécifique à la locale.

`recordsep`

Indique le séparateur d'enregistrement (ligne) à utiliser dans le mode d'affichage non aligné. La valeur par défaut est un caractère de retour chariot.

`tuples_only` (ou `t`)

Si *`valeur`* est spécifiée, elle doit valoir soit `on`, soit `off`, ce qui va activer ou désactiver le mode "tuples seulement". Si *`valeur`* est omise, la commande bascule entre la sortie normale et la sortie "tuples seulement". La sortie normale comprend des informations supplémentaires telles que les entêtes de colonnes, les titres, et différents pieds. Dans le mode "tuples seulement", seules les données de la table sont affichées.

`title [ `*`texte`*` ]`

Initialise le titre de la table pour toutes les tables affichées ensuite. Ceci peut être utilisé pour ajouter des balises de description à l'affichage. Si aucun *`valeur`* n'est donné, le titre n'est pas initialisé.

`tableattr` (or `T`)

Précise les attributs qui seront placés à l'intérieur des balises HTML `table` tag, dans le format de sortie `html`. Ceci pourrait être par exemple `cellpadding` ou `bgcolor`. Notez que vous ne

voulez probablement pas spécifier `border` car c'est pris en compte par `\pset border`. Si *`valeur`* n'est pas précisée, aucun attribut de table n'est positionné.

`pager`

Contrôle l'utilisation d'un paginateur pour les requêtes et les affichages de l'aide de psql. Si la variable d'environnement `PAGER` est configurée, la sortie est envoyée via un tube dans le programme spécifié. Sinon, une valeur par défaut dépendant de la plateforme (comme `more`) est utilisée.

Quand l'option `pager` vaut `off`, le paginateur n'est pas utilisé. Quand l'option `pager` vaut `on`, et que cela est approprié, c'est à dire quand la sortie est dirigée vers un terminal, et ne tient pas dans l'écran, le paginateur est utilisé. L'option `pager` peut également être positionnée à `always`, ce qui a pour effet d'utiliser le paginateur pour toutes les sorties terminal, que ces dernières tiennent ou non dans l'écran. `\pset pager` sans préciser *`valeur`* bascule entre les états "paginateur activé" et "paginateur désactivé".

Des exemples d'utilisation de ces différents formats sont disponibles dans la section Exemples.

Astuce

Il existe plusieurs raccourcis de commandes pour **\pset**. Voir **\a**, **\C**, **\H**, **\t**, **\T** et **\x**.

Note

C'est une erreur d'appeler **\pset** sans argument. Dans le futur, cet appel pourrait afficher le statut actuel de toutes les options d'affichage.

`\q` ou `\quit`

Quitte le programme psql. Avec un script, seule l'exécution du script est terminée.

`\qecho` *`texte`* `[ ... ]`

Cette commande est identique à **\echo** sauf que les affichages sont écrits dans le canal d'affichage des requêtes, configuré par **\o**.

`\r` ou `\reset`

Réinitialise (efface) le tampon de requêtes.

`\s [` *`nomfichier`* `]`

Sauvegarde l'historique de la ligne de commande de psql dans *`nomfichier`*. Si *`nomfichier`* est omis, l'historique est écrit sur la sortie standard (en utilisant le paginateur si nécessaire). Cette commande n'est pas disponible si psql a été construit sans le support de Readline.

`\set [` *`nom`* `[` *`valeur`* `[ ... ]]]`

Initialise la variable interne *`nom`* à *`valeur`* ou, si plus d'une valeur est donnée, à la concaténation de toutes les valeurs. Si aucun second argument n'est donné, la variable est simplement initialisée sans valeur. Pour désinitialiser une variable, utilisez la commande **\unset**.

Les noms de variables valides peuvent contenir des caractères, chiffres et tirets bas. Voir la section Variables ci-dessous pour les détails. Les noms des variables sont sensibles à la casse.

Bien que vous puissiez configurer toute variable comme vous le souhaitez, psql traite certaines variables de façon spéciale. Elles sont documentées dans la section sur les variables.

Note

Cette commande est totalement séparée de la commande SQL SET(7).

`\t`

Bascule l'affichage des en-têtes de nom de colonne en sortie et celle du bas de page indiquant le nombre de lignes. Cette commande est équivalente à `\pset tuples_only` et est fournie pour en faciliter l'accès.

\T ***options_table***

Spécifie les attributs qui seront placés dans le tag `table` pour le format de sortie HTML. Cette commande est équivalente à `\pset tableattr` *`options_table`*.

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

Sans paramètre, affiche le temps pris par chaque instruction SQL, en millisecondes, ou arrête cet affichage. Avec paramètre, force la valeur au paramètre.

\w ou \write ***nomfichier*****, \w ou \write |*****commande***

Place le tampon de requête en cours dans le fichier *`nomfichier`* ou l'envoie via un tube à la commande shell *`commande`*.

\x

Bascule le mode étendu de formatage en table. C'est équivalent à `\pset expanded`.

\z [***motif*** **]**

Liste les tables, vues et séquences avec leur droit d'accès associé. Si un *`motif`* est spécifié, seules les tables, vues et séquences dont le nom correspond au motif sont listées.

Ceci est un alias pour **\dp** (« affichage des droits »).

\! [***commande*** **]**

Lance un shell Unix séparé ou exécute la commande shell *`commande`*. Les arguments ne sont pas interprétés, le shell les voit tel quel.

\?

Affiche l'aide sur les commandes antislash.

motifs

Les différentes commandes `\d` acceptent un paramètre *`motif`* pour spécifier le(s) nom(s) d'objet à afficher. Dans le cas le plus simple, un motif est seulement le nom exact de l'objet. Les caractères à l'intérieur du motif sont normalement mis en minuscule comme pour les noms SQL ; par exemple, `\dt FOO` affichera la table nommée `foo`. Comme pour les noms SQL, placer des guillemets doubles autour d'un motif empêchera la mise en minuscule. Si vous devez inclure un guillemet double dans un motif, écrivez-le en double en accord avec les règles sur les identifiants SQL. Par exemple, `\dt "FOO""BAR"` affichera la table nommée `FOO"BAR` (et non pas `foo"bar`). Contrairement aux règles normales pour les noms SQL, vous pouvez placer des guillemets doubles simplement autour d'une partie d'un motif, par exemple `\dt FOO"FOO"BAR` affichera la table nommée `fooFOObar`.

Lorsque le paramètre *`motif`* est complètement absent, la commande `\d` affiche tous les objets visibles dans le chemin de recherche courant -- cela est équivalent à l'utilisation du motif `*`. (Un objet est dit *visible* si le schéma qui le contient est dans le chemin de recherche et qu'aucun objet de même type et même nom n'apparaît en priorité dans le chemin de recherche. Cela est équivalent à dire que l'objet peut être référencé par son nom sans préciser explicitement le schéma.) Pour voir tous les objets de la base quelle que soit leur visibilité, utilisez le motif `*.*` .

À l'intérieur d'un motif, `*` correspond à toute séquence de caractères (et aussi à aucun) alors que `?` ne correspond qu'à un seul caractère. (Cette notation est comparable à celle des motifs de nom de fichier Unix.) Par exemple, `\dt int*` affiche les tables dont le nom commence avec `int`. Mais à l'intérieur de guillemets doubles, `*` et `?` perdent leurs significations spéciales et sont donc traités directement.

Un motif qui contient un point (`.`) est interprété comme le motif d'un nom de schéma suivi par celui d'un nom d'objet. Par exemple, `\dt foo*.*bar*` affiche toutes les tables dont le nom inclut `bar` et qui sont dans des schémas dont le nom commence avec `foo`. Sans point, le motif correspond seulement avec les objets qui sont visibles dans le chemin de recherche actuel des schémas. De nouveau, un point dans des guillemets doubles perd sa signification spéciale et est traité directement.

Les utilisateurs avancés peuvent utiliser des expressions rationnelles comme par exemple les classes de caractère (`[0-9]` pour tout chiffre). Tous les caractères spéciaux d'expression rationnelle fonctionnent de la façon indiquée dans Section 9.7.3, « Expressions rationnelles POSIX », sauf pour le `.` qui est pris comme séparateur (voir ci-dessus), l'étoile (`*`) qui est transformée en l'expression rationnelle `.*` et `?` qui est transformée en `.`, et `$` qui est une correspondance littérale. Vous pouvez émuler ces caractères si besoin en écrivant `?` pour `.`, `(R+|)` pour *R*`*` et `(R|)` pour *R*`?`. `$` n'est pas nécessaire en tant que caractère d'une expression rationnelle car le motif doit correspondre au nom complet, contrairement à l'interprétation habituelle des expressions rationnelles (en d'autres termes, `$` est ajouté automatiquement à votre motif). Écrivez `*` au début et/ou à la fin si vous ne souhaitez pas que le motif soit ancré. Notez qu'à l'intérieur de guillemets doubles, tous les caractères spéciaux des expressions rationnelles perdent leur signification spéciale et sont traités directement. De plus, ces caractères sont traités littéralement dans les motifs des noms d'opérateurs (par exemple pour l'argument de `\do`).

Fonctionnalités avancées

Variables

psql fournit des fonctionnalités de substitution de variable similaire aux shells de commandes Unix. Les variables sont simplement des paires nom/valeur où la valeur peut être toute chaîne, quel que soit sa longueur. Pour initialiser des variables, utilisez la méta-commande psql **\set** :

```
basetest=> \set foo bar
```

initialise la variable `foo` avec la valeur `bar`. Pour récupérer le contenu de la variable, précédez le nom avec un caractère deux-points. Vous pouvez l'utiliser comme argument de toute commande slash :

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

Note

Les arguments de **\set** sont sujets aux même règles de substitution que les autres commandes. Du coup, vous pouvez construire des références intéressantes comme `\set :foo 'quelquechose'` et obtenir des « liens doux » ou des « variables de variables » comme, respectivement, Perl™ ou PHP™. Malheureusement (ou heureusement ?), on ne peut rien faire d'utile avec ces constructions. D'un autre côté, `\set bar :foo` est un moyen parfaitement valide de copier une variable.

Si vous appelez **\set** sans second argument, la variable est initialisée avec une chaîne vide. Pour désinitialiser (ou supprimer) une variable, utilisez la commande **\unset**.

Les noms de variables internes de psql peuvent être constitués de lettres, nombres et tirets bas dans n'importe quel ordre et autant de fois que vous le voulez. Un certain nombre de ces variables sont traitées spécialement par psql. Elles indiquent certaines options qui peuvent changer au moment de l'exécution en modifiant la valeur de la variable ou représentent un certain état de l'application. Bien que vous puissiez utiliser ces variables dans n'importe quel but, ce n'est pas recommandé car le comportement du programme pourrait devenir très rapidement vraiment étrange. Par convention, toutes les variables traitées spécialement sont uniquement composées de lettres majuscules (et peut-être aussi de chiffres et de tirets bas). Pour s'assurer d'une compatibilité maximum dans le futur, évitez l'utilisation de tels noms de variables pour vos propres besoins. Une liste de toutes les variables traitées spécialement suit.

`AUTOCOMMIT`

Si actif (`on`, valeur par défaut), chaque commande SQL est automatiquement validée si elle se termine avec succès. Pour suspendre la validation dans ce mode, vous devez saisir une commande SQL **BEGIN** ou **START TRANSACTION**. Lorsqu'elle est désactivée (`off`) ou non initialisée, les commandes SQL ne sont plus validées tant que vous ne lancez pas explicitement **COMMIT** ou **END**. Le mode sans autocommit fonctionne en lançant implicitement un **BEGIN**, juste avant toute commande qui n'est pas déjà dans un bloc de transaction et qui n'est pas elle-même un **BEGIN** ou une autre commande de contrôle de transaction, ou une commande qui ne peut pas être exécutée à l'intérieur d'un bloc de transaction (comme **VACUUM**).

Note

Dans le mode sans autocommit, vous devez annuler explicitement toute transaction échouée en saisissant **ABORT** ou **ROLLBACK**. Gardez aussi en tête que si vous sortez d'une session sans validation, votre travail est perdu.

Note

Le mode auto-commit est le comportement traditionnel de PostgreSQL™ alors que le mode sans autocommit est plus proche des spécifications SQL. Si vous préférez sans autocommit, vous pouvez le configurer dans le fichier `psqlrc` global du système ou dans votre fichier `~/.psqlrc`.

`DBNAME`

Le nom de la base de données à laquelle vous êtes actuellement connecté. Ceci est configuré à chaque fois que vous vous connectez à une base de données (ainsi qu'au lancement du programme) mais peut être désinitialisé.

`ECHO`

Si cette variable est initialisée à `all`, toutes les lignes non vides en entrées sont écrites sur la sortie standard comme elles sont lues (ceci ne s'applique pas aux lignes lues interactivement). Pour sélectionner ce comportement au lancement du programme, utilisez l'option `-a`. Si `ECHO` vaut `queries`, psql affiche chaque requête sur la sortie standard tel qu'elle est envoyée au serveur. L'option pour ceci est `-e`.

`ECHO_HIDDEN`

Quand cette variable est initialisée à `on` et qu'une commande antislash est envoyée à la base de données, la requête est d'abord affichée. De cette façon, vous pouvez étudier le fonctionnement interne de PostgreSQL™ et fournir des fonctionnalités similaires dans vos propres programmes. (Pour sélectionner ce comportement au lancement du programme, utilisez l'option `-E`.) Si vous configurez la variable avec la valeur `noexec`, les requêtes sont juste affichées mais ne sont pas réellement envoyées au serveur ni exécutées.

`ENCODING`

Le codage courant du jeu de caractères du client.

`FETCH_COUNT`

Si cette variable est un entier positif, les résultats de la requête **SELECT** sont récupérés et affichés en groupe de ce nombre de lignes, plutôt que par le comportement par défaut (récupération de l'ensemble complet des résultats avant l'affichage). Du coup, seule une petite quantité de mémoire est utilisée, quelle que soit la taille de l'ensemble des résultats. Une configuration entre 100 et 1000 est habituellement utilisée lors de l'activation de cette fonctionnalité. Gardez en tête que lors de l'utilisation de cette fonctionnalité, une requête pourrait échouer après avoir affiché quelques lignes.

Astuce

Bien que vous puissiez utiliser tout format de sortie avec cette fonctionnalité, le format par défaut, `aligned`, rend mal car chaque groupe de `FETCH_COUNT` lignes sera formaté séparément, modifiant ainsi les largeurs de colonnes suivant les lignes du groupe. Les autres formats d'affichage fonctionnent mieux.

`HISTCONTROL`

Si cette variable est configurée à `ignorespace`, les lignes commençant avec un espace n'entrent pas dans la liste de l'historique. Si elle est initialisée avec la valeur `ignoredups`, les lignes correspondant aux précédentes lignes de l'historique n'entrent pas dans la liste. Une valeur de `ignoreboth` combine les deux options. Si elle n'est pas initialisée ou si elle est configurée avec une autre valeur que celles-ci, toutes les lignes lues dans le mode interactif sont sauvegardées dans la liste de l'historique.

Note

Cette fonctionnalité a été plagiée sur Bash.

HISTFILE

Le nom du fichier utilisé pour stocker l'historique. La valeur par défaut est `~/.psql_history`. Par exemple, utiliser :

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

dans `~/.psqlrc` fera que psql maintiendra un historique séparé pour chaque base de données.

Note

Cette fonctionnalité a été plagiée sans honte à partir de Bash.

HISTSIZE

Le nombre de commandes à stocker dans l'historique des commandes. La valeur par défaut est 500.

Note

Cette fonctionnalité a été plagiée sur Bash.

HOST

L'hôte du serveur de la base de données où vous êtes actuellement connecté. Ceci est configuré à chaque fois que vous vous connectez à une base de données (ainsi qu'au lancement du programme) mais peut être désinitialisé.

IGNOREEOF

Si non initialisé, envoyer un caractère EOF (habituellement **Ctrl**+**D**) dans une session interactive de psql ferme l'application. Si elle est configurée avec une valeur numérique, ce nombre de caractères EOF est ignoré avant la fin de l'application. Si la variable est configurée mais n'a pas de valeur numérique, la valeur par défaut est de 10.

Note

Cette fonctionnalité a été plagiée sur Bash.

LASTOID

La valeur du dernier OID affecté, renvoyée à partir d'une commande **INSERT** ou **lo_import**. La validité de cette variable est seulement garantie jusqu'à l'affichage du résultat de la commande SQL suivante.

ON_ERROR_ROLLBACK

Lorsqu'il est actif (`on`), si une instruction d'un bloc de transaction génère une erreur, cette dernière est ignorée et la transaction continue. Lorsqu'il vaut `interactive`, ces erreurs sont seulement ignorées lors des sessions interactives, mais ne le sont pas lors de la lecture de scripts. Lorsqu'il vaut `off` (valeur par défaut), une instruction générant une erreur dans un bloc de transaction annule la transaction complète. Le mode on_error_rollback-on fonctionne en exécutant un **SAVEPOINT** implicite pour vous, juste avant chaque commande se trouvant dans un bloc de transaction et annule jusqu'au dernier point de sauvegarde en cas d'erreur.

ON_ERROR_STOP

Par défaut, si les scripts non interactifs rencontrent une erreur, comme une commande SQL mal formée ou une méta-commande interne, le traitement continue. Ceci a été le comportement traditionnel de psql mais il est quelque fois indésirable. Si cette variable est configurée à `on`, le traitement du script s'arrête immédiatement après une erreur. Si le script a été appelé à partir d'un autre script, il se termine de la même façon. Si le script le plus externe n'a pas été appelé à partir

d'une session interactive de psql mais plutôt en utilisant l'option `-f`, psql renvoie le code erreur 3 pour distinguer ce cas des conditions d'erreurs fatales (code d'erreur 1).

PORT

Le port du serveur de la base de données sur lequel vous êtes actuellement connecté. Ceci est configuré à chaque fois que vous vous connectez à une base de données (ainsi qu'au lancement du programme) mais peut être désinitialisé.

PROMPT1, PROMPT2, PROMPT3

Ils spécifient à quoi doit ressembler l'invite psql. Voir Invite ci-dessous.

QUIET

Configurer cette variable à `on` est équivalent à l'option `-q` en ligne de commande. Elle n'est probablement pas très utile en mode interactif.

SINGLELINE

Configurer cette variable à `on` est équivalent à l'option `-S` en ligne de commande.

SINGLESTEP

Configurer cette variable à `on` est équivalente à l'option `-s` en ligne de commande.

USER

L'utilisateur de la base de données où vous êtes actuellement connecté. Ceci est configuré à chaque fois que vous vous connectez à une base de données (ainsi qu'au lancement du programme) mais peut être désinitialisé.

VERBOSITY

Cette variable peut être configurée avec les valeurs `default`, `verbose` (bavard) ou `terse` (succinct) pour contrôler la verbosité des rapports d'erreurs.

Interpolation SQL

Une fonctionnalité utile supplémentaire des variables psql est que vous pouvez les substituer (« interpoler ») dans les instructions SQL standards. psql offre des fonctionalités spéciales pour garantir que les valeurs utilisées comme chaînes SQL litérales ou comme identifiants sont correctement échappées. La syntaxe pour interpoler une valeur sans échappement spécial est de nouveau de précéder le nom de la variable avec un caractère deux-points (`:`) :

```
basetest=> \set foo 'ma_table'
basetest=> SELECT * FROM :foo;
```

envoie alors la requête pour la table `ma_table`. Notez que cela peut être dangereux ; la valeur de la variable est copiée de façon litérale, elle peut même contenir des guillemets non fermés, ou bien des commandes backslash. Vous devez vous assurer que cela a du sens à l'endroit où vous les utilisez.

Lorsqu'une valeur doit être utilisée comme une chaîne SQL litérale ou un identifiant, il est plus sûr de s'arranger pour qu'elle soit échappée. Afin d'échapper la valeur d'une variable en tant que chaîne SQL litérale, écrivez un caractère deux-points, suivi du nom de la variable entouré par des guillemets simples. Pour échapper la valeur en tant qu'identifiant SQL, écrivez un caractère deux-points suivi du nom de la valeur entouré de guillemets doubles. L'exemple précédent peut s'écrire de façon plus sûre ainsi :

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

L'interpolation de variable n'est pas effectuée dans les entités SQL entourées de guillemets. SQL

Une utilisation possible de ce mécanisme est de copier le contenu d'un fichier dans une colonne d'une table. Tout d'abord, chargez le fichier dans une variable puis procédez ainsi :

```
basetest=> \set contenu `cat mon_fichier.txt`
basetest=> INSERT INTO ma_table VALUES (:'contenu');
```

(Notez que cela ne fonctionnera par si le fichier `mon_fichier.txt` contient des octets nuls. psql ne gère pas les octets nuls inclus dans les valeurs de variable.)

Comme les caractères deux-points peuvent légitimement apparaître dans les commandes SQL, une tentative apparente d'interpolation (comme `:nom`, `:'nom'`, or `:"nom"`) n'est pas modifiée, sauf si la variable nommée est actuellement positionnée. Dans tous les cas, vous pouvez échapper un caractère deux-points avec un backslash pour le protéger des substitutions. (La syntaxe deux-points pour les variables est du SQL standard pour les langages de requête embarqués, comme ECPG. La syntaxe avec les deux-points pour les tranches de tableau et les conversions de types sont des extensions PostgreSQL™ extensions, d'où le conflit. La syntaxe avec le caractère deux-points pour échapper la valeur d'une variable en tant que chaîne SQL litérale ou identifiant est une extension psql .)

Invite

Les invites psql peuvent être personnalisées suivant vos préférences. Les trois variables `PROMPT1`, `PROMPT2` et `PROMPT3` contiennent des chaînes et des séquences d'échappement spéciales décrivant l'apparence de l'invite. L'invite 1 est l'invite normale qui est lancée quand psql réclame une nouvelle commande. L'invite 2 est lancée lorsqu'une saisie supplémentaire est attendue lors de la saisie de la commande parce que la commande n'a pas été terminée avec un point-virgule ou parce qu'un guillemet n'a pas été fermé. L'invite 3 est lancée lorsque vous exécutez une commande SQL **COPY** et que vous devez saisir les valeurs des lignes sur le terminal.

La valeur de la variable prompt sélectionnée est affichée littéralement sauf si un signe pourcentage (`%`) est rencontré. Suivant le prochain caractère, certains autres textes sont substitués. Les substitutions définies sont :

`%M`

Le nom complet de l'hôte (avec le nom du domaine) du serveur de la base de données ou `[local]` si la connexion est établie via une socket de domaine Unix ou `[local:`*`/répertoire/nom`*`]`, si la socket de domaine Unix n'est pas dans l'emplacement par défaut défini à la compilation.

`%m`

Le nom de l'hôte du serveur de la base de données, tronqué au premier point ou `[local]` si la connexion se fait via une socket de domaine Unix.

`%>`

Le numéro de port sur lequel le serveur de la base de données écoute.

`%n`

Le nom d'utilisateur de la session. (L'expansion de cette valeur peut changer pendant une session après une commande **SET SESSION AUTHORIZATION**.)

`%/`

Le nom de la base de données courante.

`%~`

Comme `%/` mais l'affichage est un ~ (tilde) si la base de données est votre base de données par défaut.

`%#`

Si l'utilisateur de la session est un superutilisateur, alors un # sinon un >. (L'expansion de cette valeur peut changer durant une session après une commande **SET SESSION AUTHORIZATION**.)

`%R`

Pour l'invite 1, affiche normalement = mais affiche ^ si on est en mode simple ligne et `!` si la session est déconnectée de la base de données (ce qui peut arriver si **\connect** échoue). Pour l'invite 2, la séquence est remplacée par `-`, `*`, un simple guillemet, un double ou un signe dollar, suivant si psql attend une saisie supplémentaire parce que la commande n'est pas terminée, parce que vous êtes à l'intérieur d'un commentaire `/* ... */`, ou parce que vous n'avez pas terminé un guillemet ou une chaîne échappée avec des dollars. Pour l'invite 3, la séquence ne produit rien.

`%x`

État de la Transaction : une chaîne vide lorsque vous n'êtes pas dans un bloc de transaction ou `*` si vous vous y trouvez, ou `!` si vous êtes dans une transaction échouée, ou enfin `?` lorsque l'état de la transaction est indéterminé (par exemple à cause d'une rupture de la connexion).

`%`*`chiffres`*

Le caractère avec ce code numérique est substitué.

`%:`*`nom`*`:`

La valeur de la variable *`nom`* de psql. Voir la section Variables pour les détails.

`%`` *`commande`* ``

la sortie de la *`commande`*, similaire à la substitution par « guillemets inverse » classique.

`%[ ... %]`

Les invites peuvent contenir des caractères de contrôle du terminal qui, par exemple, modifient la couleur, le fond ou le style du texte de l'invite, ou modifient le titre de la fenêtre du terminal. Pour que les fonctionnalités d'édition de ligne de Readline fonctionnent correctement, les caractères de contrôle non affichables doivent être indiqués comme invisibles en les entourant avec `%[` et `%]`. Des pairs multiples de ceux-ci pourraient survenir à l'intérieur de l'invite. Par exemple :

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

a pour résultat une invite en gras (`1;`), jaune sur noir (`33;40`) sur les terminaux compatibles VT100.

Pour insérer un pourcentage dans votre invite, écrivez `%%`. Les invites par défaut sont `'%/%R%# '` pour les invites 1 et 2 et `'>> '` pour l'invite 3.

Note

Cette fonctionnalité a été plagiée sur tcsh.

Édition de la ligne de commande

psql supporte la bibliothèque Readline pour une édition et une recherche simplifiée et conviviale de la ligne de commande. L'historique des commandes est automatiquement sauvegardé lorsque psql quitte et est rechargé quand psql est lancé. La complétion par tabulation est aussi supportée bien que la logique de complétion n'ait pas la prétention d'être un analyseur SQL. Si pour quelques raisons que ce soit, vous n'aimez pas la complétion par tabulation, vous pouvez la désactiver en plaçant ceci dans un fichier nommé `.inputrc` de votre répertoire personnel :

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

(Ceci n'est pas une fonctionnalité psql mais Readline. Lisez sa documentation pour plus de détails.)

Environnement

`COLUMNS`

Si `\pset columns` vaut zéro, contrôle la largeur pour le format `wrapped` et la largeur pour déterminer si une sortie large a besoin du pager.

`PAGER`

Si les résultats d'une requête ne tiennent pas sur l'écran, ils sont envoyés via un tube sur cette commande. Les valeurs typiques sont `more` ou `less`. La valeur par défaut dépend de la plateforme. L'utilisation du paginateur peut être désactivée en utilisant la commande **\pset**.

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

Paramètres de connexion par défaut (voir Section 31.13, « Variables d'environnement »).

`PSQL_EDITOR`, `EDITOR`, `VISUAL`

Éditeur utilisé par la commande **\e**. Les variables sont examinées dans l'ordre donné ; la première initialisée est utilisée.

SHELL

Commande exécutée par la commande **\!**.

TMPDIR

Répertoire pour stocker des fichiers temporaires. La valeur par défaut est `/tmp`.

Cet outil, comme la plupart des autres outils PostgreSQL™, utilise aussi les variables d'environnement supportées par la bibliothèque libpq (voir Section 31.13, « Variables d'environnement »).

Fichiers

- Sauf si une option `-X` ou `-c` est fournie, psql tente de lire et exécuter les commandes provenant du fichier global au système `psqlrc` ou du fichier utilisateur `~/.psqlrc` avant de démarrer. (Sur Windows, le fichier de démarrage de l'utilisateur est nommé `%APPDATA%\postgresql\psqlrc.conf`.) Voir *`PREFIX`*`/share/psqlrc.sample` pour plus d'informations sur la configuration du fichier global au système. Il pourrait être utilisé pour configurer le client et le serveur à votre goût (en utilisant les commandes **\set** et **SET**).

- À la fois le fichier `psqlrc` global au système et le fichier `~/.psqlrc` de l'utilisateur peuvent être créés en étant spécifiques à une version si vous leur ajouter un tiret et le numéro de version de `~/.psqlrc-9.0.23`. Un fichier correspondant à une version spécifique est préféré à un fichier sans indication de version.

- L'historique de la ligne de commandes est stocké dans le fichier `~/.psql_history` ou `%APPDATA%\postgresql\psql_history` sur Windows.

Notes

- Dans une vie précédente, psql permettait au premier argument d'une commande antislash à une seule lettre de commencer directement après la commande, sans espace séparateur. À partir de PostgreSQL™ 8.4, ce n'est plus autorisé.

- Le fonctionnement de psql n'est garanti qu'avec des serveurs de même version. Cela ne signifie pas que d'autres combinaisons vont complètement échouer, mais des problèmes subtils, voire moins subtils, pourraient apparaître. Les méta-commandes sont particulièrements fragiles si le serveur est d'une version plus récente que psql lui-même. Toutefois, les commandes backslash de la famille `\d` devraient fonctionner avec des serveurs 7.4 jusqu'à la version courante, même si pas nécessairement avec des serveurs plus récents que psql lui-même.

Notes pour les utilisateurs sous Windows

psql est construit comme une « application de type console ». Comme les fenêtres console de windows utilisent un codage différent du reste du système, vous devez avoir une attention particulière lors de l'utilisation de caractères sur 8 bits à l'intérieur de psql. Si psql détecte une page de code problématique, il vous avertira au lancement. Pour modifier la page de code de la console, deux étapes sont nécessaires :

- Configurez la page code en saisissant **`cmd.exe /c chcp 1252`**. (1252 est une page code appropriée pour l'Allemagne ; remplacez-la par votre valeur.) Si vous utilisez Cygwin, vous pouvez placer cette commande dans `/etc/profile`.

- Configurez la police de la console par `Lucida Console` parce que la police raster ne fonctionne pas avec la page de code ANSI.

Exemples

Le premier exemple montre comment envoyer une commande sur plusieurs lignes d'entrée. Notez le changement de l'invite :

```
basetest=> CREATE TABLE ma_table (
basetest(>  premier integer not NULL default 0,
basetest(>  second text)
basetest-> ;
CREATE TABLE
```

Maintenant, regardons la définition de la table :

```
basetest=> \d ma_table
             Table "ma_table"
 Attribute |  Type   |      Modifier
-----------+---------+--------------------
 premier   | integer | not null default 0
 second    | text    |
```

Maintenant, changeons l'invite par quelque chose de plus intéressant :

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

Supposons que nous avons rempli la table de données et que nous voulons les regarder :

```
peter@localhost basetest=> SELECT * FROM ma_table;
 premier | second
---------+--------
       1 | one
       2 | two
       3 | three
       4 | four
(4 rows)
```

Vous pouvez afficher cette table de façon différente en utilisant la commande **\pset** :

```
peter@localhost basetest=> \pset border 2
Border style is 2.
peter@localhost basetest=> SELECT * FROM ma_table;
+---------+--------+
| premier | second |
+---------+--------+
|       1 | one    |
|       2 | two    |
|       3 | three  |
|       4 | four   |
+---------+--------+
(4 rows)

peter@localhost basetest=> \pset border 0
Border style is 0.
peter@localhost basetest=> SELECT * FROM ma_table;
premier second
------- ------
      1 one
      2 two
      3 three
      4 four
(4 rows)

peter@localhost basetest=> \pset border 1
Border style is 1.
peter@localhost basetest=> \pset format unaligned
Output format is unaligned.
peter@localhost basetest=> \pset fieldsep ","
Field separator is ",".
peter@localhost basetest=> \pset tuples_only
Showing only tuples.
peter@localhost basetest=> SELECT second, first FROM
ma_table;
one,1
two,2
three,3
four,4
```

Vous pouvez aussi utiliser les commandes courtes :

```
peter@localhost basetest=> \a \t \x
Output format is aligned.
Tuples only is off.
Expanded display is on.
peter@localhost basetest=> SELECT * FROM ma_table;
-[ RECORD 1 ]-
```

```
first  | 1
second | one
-[ RECORD 2 ]-
first  | 2
second | two
-[ RECORD 3 ]-
first  | 3
second | three
-[ RECORD 4 ]-
first  | 4
second | four
```