pg_staging

Dimitri Fontaine

November, 7th 2009

How to manage your staging environments?

About backups

The idea of `pg_staging` is to maintain a *staging* environment from production backups. If you don't have one, this tool will not do any good for you, but you have bigger problems than that. Supported backups are:

- `pg_dump`
- *PITR archives* (ongoing work)

About backups

The idea of `pg_staging` is to maintain a *staging* environment from production backups. If you don't have one, this tool will not do any good for you, but you have bigger problems than that. Supported backups are:

- `pg_dump`
- *PITR archives* (ongoing work)

About backups

The idea of `pg_staging` is to maintain a *staging* environment from production backups. If you don't have one, this tool will not do any good for you, but you have bigger problems than that. Supported backups are:

- `pg_dump`
- *PITR archives* (ongoing work)

About backups

When developping `pg_staging`, the aim is to manage a staging environment with:

- a dev env, with new code and old database containing developpers test data
- a prelive env with code to get in production and the most recent possible data from production

About backups

When developping `pg_staging`, the aim is to manage a staging environment with:

- a dev env, with new code and old database containing developpers test data
- a prelive env with code to get in production and the most recent possible data from production

About backups

When developping `pg_staging`, the aim is to manage a staging environment with:

- a dev env, with new code and old database containing developpers test data
- a prelive env with code to get in production and the most recent possible data from production

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Design

What do we want the tool to do?

- easily restore production database on staging env
- filtering out (*historical*) data we don't need in staging
- allow to restore more than one version of production database
- allow to easily switch from one version to the other
- offer interactive console usage and be cron friendly too

Easy restore with filtering

The `restore` command will `createdb`, *fetch* the wanted backup, *filter* the dump catalog, `pg_restore` selected data then optionaly switch the staging env to this new database.

Example

```
schemas          = public, payment, utils, jdb, operations, s
schemas_nodata   = sessions, archives
```

Easy restore with filtering

The `restore` command will `createdb`, *fetch* the wanted backup, *filter* the dump catalog, `pg_restore` selected data then optionaly switch the staging env to this new database.

Example

```
schemas          = public, payment, utils, jdb, operations, s
schemas_nodata   = sessions, archives
```

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Dependancies

The following tools are used by `pg_staging`:

- `apache` to serve the backups
- `pgbouncer` for database switching
- `postgresql-client-8.x` for dump & restore
- `staging-client.sh`
- non-interactive `ssh`
- `python`

Switching databases with pgbouncer

pgbouncer is able to provide “virtual” database:

Example

```
pg_staging> pgbouncer some_db.dev
            some_db      some_db_20091029 :5432
    some_db_20090717     some_db_20090717 :5432
    some_db_20091029     some_db_20091029 :5432
```

So pg_staging is able to switch staging database without editing application connection strings.

Switching databases with pgbouncer

pgbouncer is able to provide "virtual" database:

Example

```
pg_staging> pgbouncer some_db.dev
                 some_db      some_db_20091029 :5432
     some_db_20090717         some_db_20090717 :5432
     some_db_20091029         some_db_20091029 :5432
```

So pg_staging is able to switch staging database without editing application connection strings.

Switching databases with pgbouncer

pgbouncer is able to provide "virtual" database:

Example

```
pg_staging> pgbouncer some_db.dev
                 some_db      some_db_20091029 :5432
     some_db_20090717         some_db_20090717 :5432
     some_db_20091029         some_db_20091029 :5432
```

So pg_staging is able to switch staging database without editing application connection strings.

Filtering objects from dump files

Using `pg_restore` options -l and -L, `pg_staging` can filter out objects at restore time, per schema.

A 2-pass parsing is done, so that even triggers depending on functions that are in the given excluded schemas are filtered out.

See commands `catalog` & `triggers` to check for yourself the catalog that will get used.

Filtering objects from dump files

Using `pg_restore` options -l and -L, `pg_staging` can filter out objects at restore time, per schema.

A 2-pass parsing is done, so that even triggers depending on functions that are in the given excluded schemas are filtered out.

See commands `catalog` & `triggers` to check for yourself the catalog that will get used.

Cluster globals, init command

To be able to restore you need to *create* the roles. Store next to your backups `pg_dumpall -globals-only` and edit your setup accordingly.

Example

```
dumpall_url = /clusters/myserver/8.3-main.globals.sql
```

Cluster globals, init command

To be able to restore you need to *create* the roles. Store next to your backups `pg_dumpall -globals-only` and edit your setup accordingly.

Example

```
dumpall_url = /clusters/myserver/8.3-main.globals.sql
```

custom SQL

Setup a `sql_path` entry in your configuration file then `restore` command will play all files in there:

Example

```
psql ... -f pre/*.sql
pg_restore
psql ... -f post/*.sql
```

custom SQL

Setup a `sql_path` entry in your configuration file then `restore` command will play all files in there:

Example

```
psql ... -f pre/*.sql
pg_restore
psql ... -f post/*.sql
```

pg_dump support

`pg_staging` also provides a `dump` command for producing a local dump from the remote configured server.

Use `redump` if you want to overwrite a previous existing dump file.

pg_dump support

`pg_staging` also provides a `dump` command for producing a local dump from the remote configured server.

Use `redump` if you want to overwrite a previous existing dump file.

londiste support

Given a *replication.ini* setup (to be documented soon, but we actually use the feature), `pg_staging` is able to configure *londiste* and the associated *pgq ticker*.

It will also skip restoring tables we are a subscriber of, see `nodata` command.

londiste support

Given a *replication.ini* setup (to be documented soon, but we actually use the feature), `pg_staging` is able to configure *londiste* and the associated *pgq ticker*.

It will also skip restoring tables we are a subscriber of, see `nodata` command.

replication.ini example

```
[DEFAULT]
pgq_lazy_fetch  = 500

[some_db_ticker.dev]
job_name = pgq_some_db
host     = bdd1.service.dev
db       = dbname=some_db user=postgres port=5432 host=127.0
```

replication.ini example

```
[payment_to_reporting.dev]
host              = bdd2.service.dev
ticker            = some_db_ticker.dev
pgq_queue_name    = service_to_reporting
provider          = some_service.dev
subscriber        = some_reporting.dev
provider_db       = dbname=some_service host=localhost port=64
subscriber_db     = dbname=some_reporting host=reporting.servi
provides = schema.table1 schema.table2 other_schema.other_ta
```

pg_staging notions

Some generics about `pg_staging` usage:

- *commands* are the same in console and CLI
- *config* is made of `.INI` files
- *dbname* refers to .INI section
- *backup date* defaults to *today*

pg_staging notions

Some generics about `pg_staging` usage:

- *commands* are the same in console and CLI
- *config* is made of `.INI` files
- *dbname* refers to .INI section
- *backup date* defaults to *today*

pg_staging notions

Some generics about `pg_staging` usage:

- *commands* are the same in console and CLI
- *config* is made of `.INI` files
- *dbname* refers to .INI section
- *backup date* defaults to *today*

pg_staging notions

Some generics about `pg_staging` usage:

- *commands* are the same in console and CLI
- *config* is made of `.INI` files
- *dbname* refers to .INI section
- *backup date* defaults to *today*

pg_staging notions

Some generics about `pg_staging` usage:

- *commands* are the same in console and CLI
- *config* is made of `.INI` files
- *dbname* refers to .INI section
- *backup date* defaults to *today*

main commands

```
pg_staging> help
  commands provide a user friendly listing of commands
      init <dbname>
   restore <dbname> restore a database
      drop <dbname> drop given database
      dump <dbname> dump a database
    redump dump a database, overwriting the pre-existing dum
      pitr <dbname> <time|xid> value
```

listing resources

```
pg_staging> help
        ...
 databases list configured databases
   backups list available backups for a given database
    dbsize show given database size
   dbsizes show dbsize for all databases of a dbname sectio
      psql launch a psql connection to the given configured
```

and some more

```
pg_staging> help
        ...
      fetch <dbname> [date]
       load <dbname> <dumpfile>

       show show given database setting current value
        get <dbname> <option> print the current value of [dbn
        set <dbname> <option> <value> for current session onl
```

managing services

```
pg_staging> help
        ...
 pgbouncer list configured pgbouncer databases
     pause pause <dbname> [date] (when no date given, not ex
    resume resume <dbname> [date] (when no date given, not e
    switch <dbname> <bdate> switch default pgbouncer config

  londiste prepare londiste files for providers of given dbn
    status show status of given service ...
     start start given service depending on its configuratio
      stop stop given service depending on its configuration
   restart restart given service depending on its configurat
```

internals you might have a use for

```
pg_staging> help
        ...
    catalog <dbname> [dump] print catalog for dbname, edited
   triggers <dbname> [dump] print triggers procedures for dbn
     nodata list tables to restore without their data
     presql <dbname> [date]
    postsql <dbname> [date]
search_path alter database <dbname> set search_path
```

Interactive or not?

Remember the “design” slide, we want to have both an interactive tool and a script friendly (think `cron`) one.

Example

```
#  pg_staging restore mydb today
#  pg_staging < foo.pgs
#  pg_staging
Welcome to pg_staging 0.7.
pg_staging>
```

Interactive or not?

Remember the "design" slide, we want to have both an interactive tool and a script friendly (think `cron`) one.

Example

```
#  pg_staging restore mydb today
#  pg_staging < foo.pgs
#  pg_staging
Welcome to pg_staging 0.7.
pg_staging>
```

pgfoundry or github...

`http://pgfoundry.org/projects/pgstaging` will host the releases when they happen.

Example

```
# pg_staging.py --version
pg_staging 0.7
```

`http://github.com/dimitri/pg_staging` is hosting the code, along with the debian packaging. Go `git clone` and try it, we use it about daily.

pgfoundry or github...

`http://pgfoundry.org/projects/pgstaging` will host the releases when they happen.

Example

```
# pg_staging.py --version
pg_staging 0.7
```

`http://github.com/dimitri/pg_staging` is hosting the code, along with the debian packaging. Go `git clone` and try it, we use it about daily.

pgfoundry or github...

`http://pgfoundry.org/projects/pgstaging` will host the releases when they happen.

Example

```
# pg_staging.py --version
pg_staging 0.7
```

`http://github.com/dimitri/pg_staging` is hosting the code, along with the debian packaging. Go `git clone` and try it, we use it about daily.

Any question?

Now is the time to ask!

If you want to leave feedback, consider visiting
`http://2009.pgday.eu/feedback`

Any question?

Now is the time to ask!

If you want to leave feedback, consider visiting
`http://2009.pgday.eu/feedback`