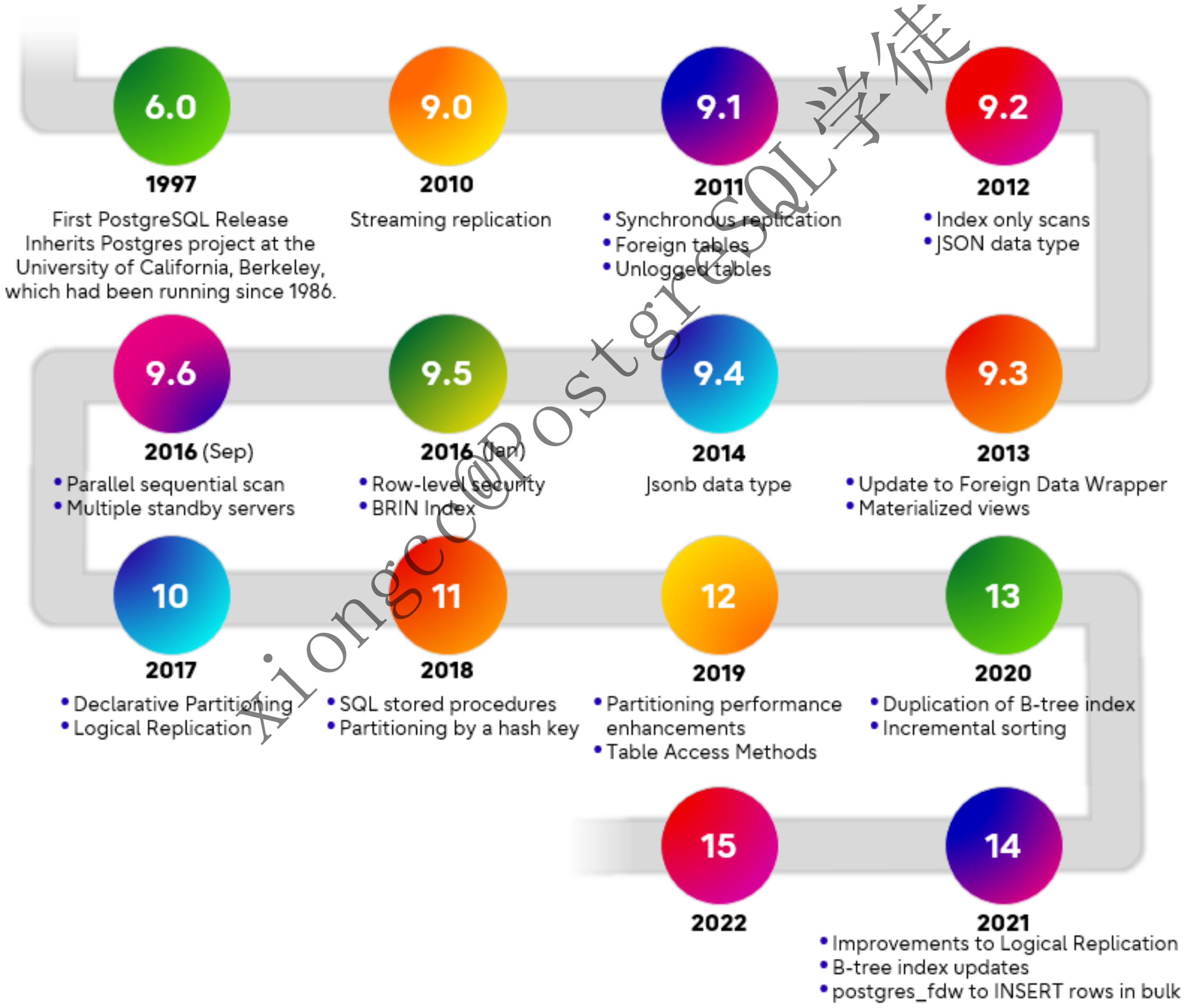
前言

版本升级对于运维人员和开发人员来说，是经常打交道的一件事，除了引入新的功能和特性之外，另外一点，便是对已知的bug和安全漏洞进行修复。我们知道，PostgreSQL的版本迭代是很快的，三个月一个小版本，一年一个大版本，每个大版本几乎都有重磅特性，如v9.4 支持了jsonb，v9.6 支持了并行，v10 支持了逻辑复制和声明式分区、v11 支持了JIT、存储过程等等，除了这些，对于运维人员和DBA来说，还会引入很多管理特性，比如 v14 引入的idle_session_timeout，就再也不需要配合 pg_timeout 插件或者写一堆复杂的定时kill脚本了，也不用担忧被大量的空闲连接和没有经验的连接池配置将连接数打爆了。

所以数据库版本升级是一件十分重要的事，避免踩到已知的bug，导致业务的不稳定。如果想了解个多的新特性或者每个特性的详细信息，请参考官网：https://www.postgresql.org/docs/current/static/release.html，每一个新特性在特定场景下，都会对数据库和应用程序的性能带来质的飞跃，都会极大的节省开发及运维人员的时间成本、人力成本，所以我们为什么不站在巨人的肩膀上前行呢?



6.0
1997
First PostgreSQL Release Inherits Postgres project at the University of California, Berkeley, which had been running since 1986.
9.0
2010
Streaming replication
9.1
2011
• Synchronous replication
• Foreign tables
• Unlogged tables
9.2
2012
• Index only scans
• JSON data type
9.6
2016 (Sep)
• Parallel sequential scan
• Multiple standby servers
9.5
2016 (Jan)
• Row-level security
• BRIN Index
9.4
2014
Jsonb data type
9.3
2013
• Update to Foreign Data Wrapper
• Materialized views
10
2017
• Declarative Partitioning
• Logical Replication
11
2018
• SQL stored procedures
• Partitioning by a hash key
12
2019
• Partitioning performance enhancements
• Table Access Methods
13
2020
• Duplication of B-tree index
• Incremental sorting
15
2022
14
2021
• Improvements to Logical Replication
• B-tree index updates
• postgres_fdw to INSERT rows in bulk

EOL

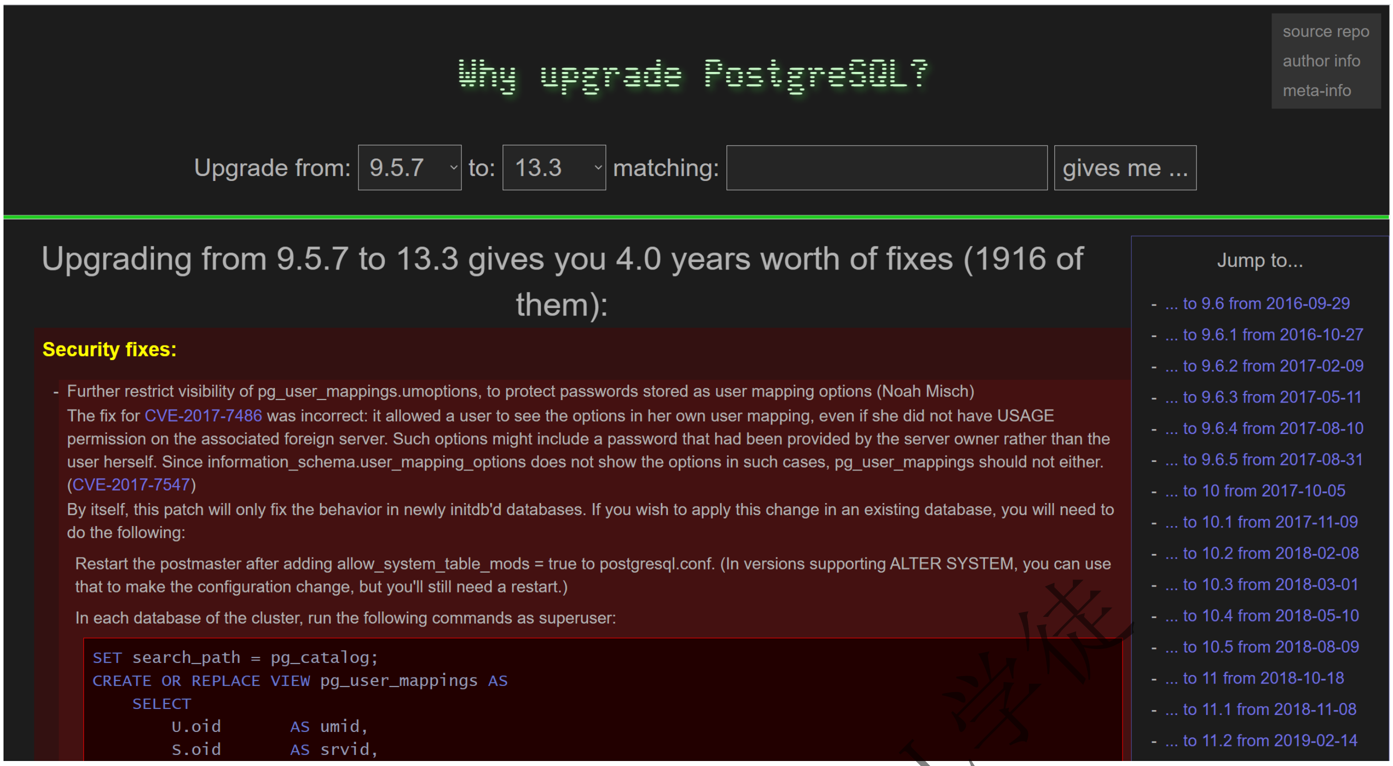
PostgreSQL 社区成员每年只会同时维护5个主版本，也就是说每个数据库版本都是有生命周期的，也即EOL，End of life

Version	Current minor	Supported	First Release	Final Release
14	14.1	Yes	September 30, 2021	November 12, 2026
13	13.5	Yes	September 24, 2020	November 13, 2025
12	12.9	Yes	October 3, 2019	November 14, 2024
11	11.14	Yes	October 18, 2018	November 9, 2023
10	10.19	Yes	October 5, 2017	November 10, 2022
9.6	9.6.24	No	September 29, 2016	November 11, 2021
9.5	9.5.25	No	January 7, 2016	February 11, 2021

所以从今年 2 月份开始，对于 9.5 以前的版本数据库不再进行维护，当然你依然可以在官网上下载 9.5 以前的版本使用，但是如果在使用过程中出现了bug，那么社区成员只会在维护的版本中修复，9.5 以前的版本不会进行任何修改，所以为了避免再次踩到低版本的bug，升级数据库还是很有必要的。

快速对比

前文提到了升级的必要性，但是可以看到，PostgreSQL的版本太多了，如何高效对比呢？比如我从远古版本 9.5 升级到最新的13，中间过了好几年，不用想肯定数不尽的issue list，当然一股脑升上去也没啥问题，毕竟新版本总是好用的多，但是对于我们运维人员来说，还是需要知其然，神器来了：https://why-upgrade.depesz.com/show?from=9.4&to=9.6.6&keywords=，看看效果，假如我要从9.5 - > 13，直接一步到位，可以看到修复了1916个bug，太好用了！感谢Hubert Lubaczewski大师。



Upgrading from 9.5.7 to 13.3 gives you 4.0 years worth of fixes (1916 of them):

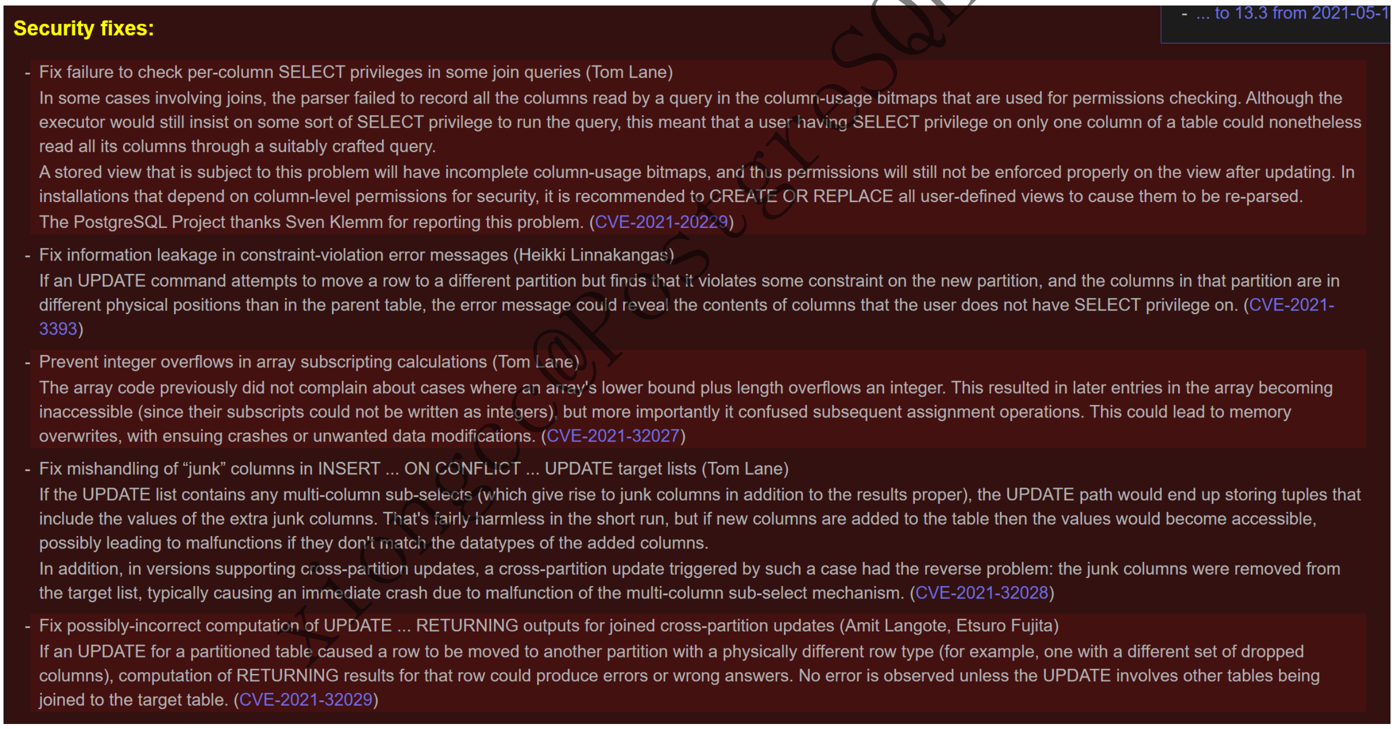
Jump to...

- ... to 9.6 from 2016-09-29
- ... to 9.6.1 from 2016-10-27
- ... to 9.6.2 from 2017-02-09
- ... to 9.6.3 from 2017-05-11
- ... to 9.6.4 from 2017-08-10
- ... to 9.6.5 from 2017-08-31
- ... to 10 from 2017-10-05
- ... to 10.1 from 2017-11-09
- ... to 10.2 from 2018-02-08
- ... to 10.3 from 2018-03-01
- ... to 10.4 from 2018-05-10
- ... to 10.5 from 2018-08-09
- ... to 11 from 2018-10-18
- ... to 11.1 from 2018-11-08
- ... to 11.2 from 2019-02-14

Security fixes:

- Further restrict visibility of pg_user_mappings.umoptions, to protect passwords stored as user mapping options (Noah Misch)
  The fix for CVE-2017-7486 was incorrect: it allowed a user to see the options in her own user mapping, even if she did not have USAGE permission on the associated foreign server. Such options might include a password that had been provided by the server owner rather than the user herself. Since information_schema.user_mapping_options does not show the options in such cases, pg_user_mappings should not either. (CVE-2017-7547)
  By itself, this patch will only fix the behavior in newly initdb'd databases. If you wish to apply this change in an existing database, you will need to do the following:
  Restart the postmaster after adding allow_system_table_mods = true to postgresql.conf. (In versions supporting ALTER SYSTEM, you can use that to make the configuration change, but you'll still need a restart.)
  In each database of the cluster, run the following commands as superuser:

```
SET search_path = pg_catalog;
CREATE OR REPLACE VIEW pg_user_mappings AS
    SELECT
        U.oid       AS umid,
        S.oid       AS srvid,
```



Security fixes:

- Fix failure to check per-column SELECT privileges in some join queries (Tom Lane)
  In some cases involving joins, the parser failed to record all the columns read by a query in the column-usage bitmaps that are used for permissions checking. Although the executor would still insist on some sort of SELECT privilege to run the query, this meant that a user having SELECT privilege on only one column of a table could nonetheless read all its columns through a suitably crafted query.
  A stored view that is subject to this problem will have incomplete column-usage bitmaps, and thus permissions will still not be enforced properly on the view after updating. In installations that depend on column-level permissions for security, it is recommended to CREATE OR REPLACE all user-defined views to cause them to be re-parsed.
  The PostgreSQL Project thanks Sven Klemm for reporting this problem. (CVE-2021-20229)
- Fix information leakage in constraint-violation error messages (Heikki Linnakangas)
  If an UPDATE command attempts to move a row to a different partition but finds that it violates some constraint on the new partition, and the columns in that partition are in different physical positions than in the parent table, the error message could reveal the contents of columns that the user does not have SELECT privilege on. (CVE-2021-3393)
- Prevent integer overflows in array subscripting calculations (Tom Lane)
  The array code previously did not complain about cases where an array's lower bound plus length overflows an integer. This resulted in later entries in the array becoming inaccessible (since their subscripts could not be written as integers), but more importantly it confused subsequent assignment operations. This could lead to memory overwrites, with ensuing crashes or unwanted data modifications. (CVE-2021-32027)
- Fix mishandling of “junk” columns in INSERT ... ON CONFLICT ... UPDATE target lists (Tom Lane)
  If the UPDATE list contains any multi-column sub-selects (which give rise to junk columns in addition to the results proper), the UPDATE path would end up storing tuples that include the values of the extra junk columns. That's fairly harmless in the short run, but if new columns are added to the table then the values would become accessible, possibly leading to malfunctions if they don't match the datatypes of the added columns.
  In addition, in versions supporting cross-partition updates, a cross-partition update triggered by such a case had the reverse problem: the junk columns were removed from the target list, typically causing an immediate crash due to malfunction of the multi-column sub-select mechanism. (CVE-2021-32028)
- Fix possibly-incorrect computation of UPDATE ... RETURNING outputs for joined cross-partition updates (Amit Langote, Etsuro Fujita)
  If an UPDATE for a partitioned table caused a row to be moved to another partition with a physically different row type (for example, one with a different set of dropped columns), computation of RETURNING results for that row could produce errors or wrong answers. No error is observed unless the UPDATE involves other tables being joined to the target table. (CVE-2021-32029)

升级方式

数据库升级分为两种，一种是小版本迭代升级，另一种是主版本升级。PostgreSQL 版本号由主要版本和次要版本组成。例如，PostgreSQL 12.4 中的 12 是主要版本，4 是次要版本；PostgreSQL 10.0 之前的版本由 3 个数字组成，例如 9.6.19，其中 9.6 是主要版本，19 是次要版本。

> Starting with PostgreSQL 10, a major version is indicated by increasing the first part of the version, e.g. 10 to 11. Before PostgreSQL 10, a major version was indicated by increasing either the first or second part of the version number, e.g. 9.5 to 9.6.

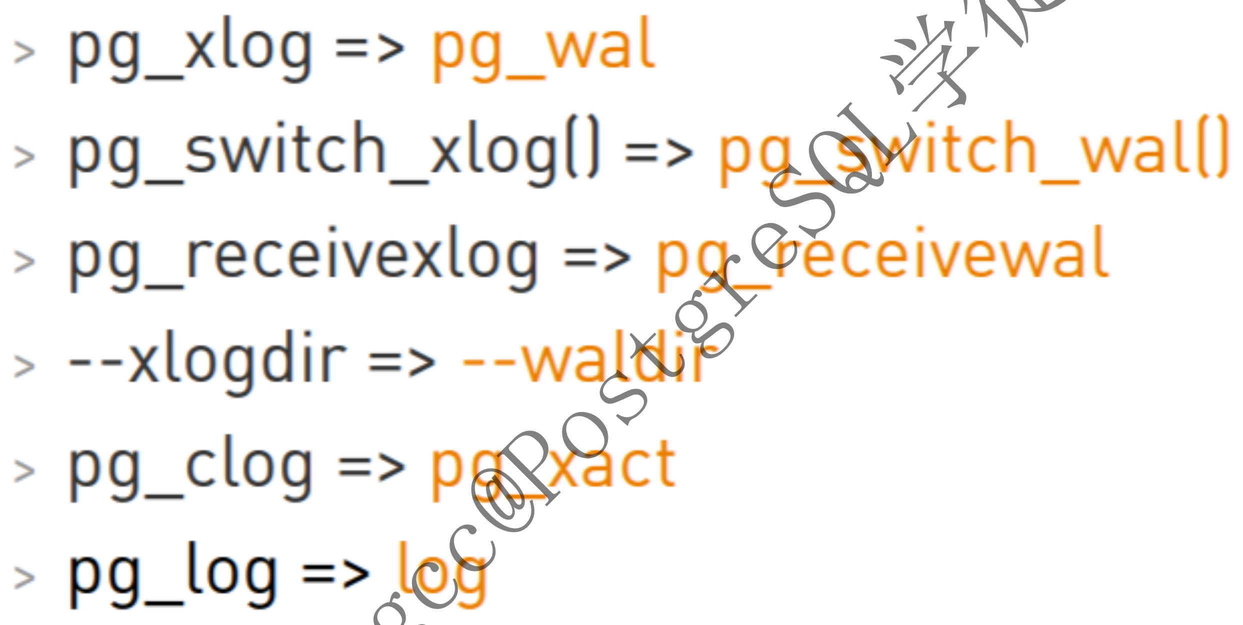
> Minor releases are numbered by increasing the last part of the version number. Beginning with PostgreSQL 10, this is the second part of the version number, e.g. 10.0 to 10.1; for older versions this is the third part of the version number, e.g. 9.5.3 to 9.5.4.

小版本升级

小版本升级不会改变内部的存储格式，因此总是和大版本兼容。例如，PostgreSQL 12.4 和 PostgreSQL 12.0 以及后续的 PostgreSQL 12.x 兼容。**对于这些兼容版本的升级非常简单，只需要关闭数据库服务，安装替换二进制的可执行文件，重新启动服务即可。**

大版本升级

接下来，我们主要讨论 PostgreSQL 的跨版本升级问题，例如从 PostgreSQL 12.x 升级到 PostgreSQL 13.x。大版本的升级可能会修改内部数据的存储格式，因此需要执行额外的操作。比如PostgreSQL 10，将 log 改成了 wal，主要是为了防止某些新手将 xlog、clog 认为是日志文件，误删了，如果还没有备份那么就GG了。



- pg_xlog => pg_wal
- pg_switch_xlog() => pg_switch_wal()
- pg_receivexlog => pg_receivewal
- --xlogdir => --waldir
- pg_clog => pg_xact
- pg_log => log

甚至PostgreSQL为了数据的安全性，高版本还不能直接使用低版本创建的数据目录，会在日志中打印相关的错误信息。

```
[postgres@xiongcc ~]$ pg_ctl --version
pg_ctl (PostgreSQL) 13.2
[postgres@xiongcc ~]$ pg_ctl -D 12data/ start
waiting for server to start....2021-07-06 14:49:34.981 CST [15232] FATAL:  database
files are incompatible with server
2021-07-06 14:49:34.981 CST [15232] DETAIL:  The data directory was initialized by
PostgreSQL version 12, which is not compatible with this version 13.2.
 stopped waiting
pg_ctl: could not start server
Examine the log output.
```

官方提供三种大版本升级方案

1. Upgrading Data via pg_dumpall，使用 pg_dumpall / pg_restore 进行升级

2. Upgrading Data via pg_upgrade，使用 pg_upgrade 进行升级
3. Upgrading Data via Replication，使用逻辑复制进行升级

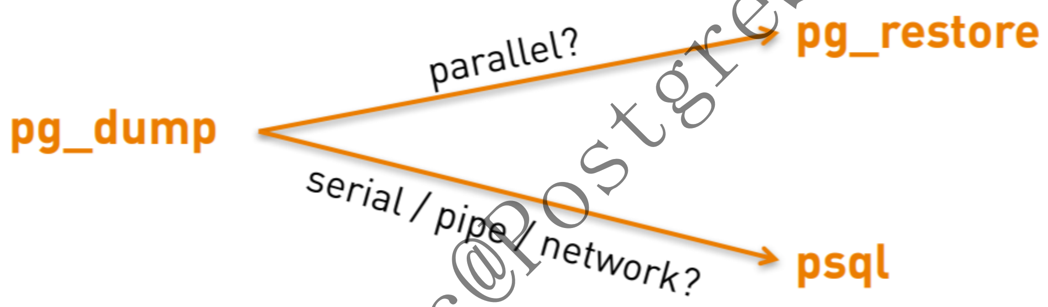
当然，还可以进行一下引申：

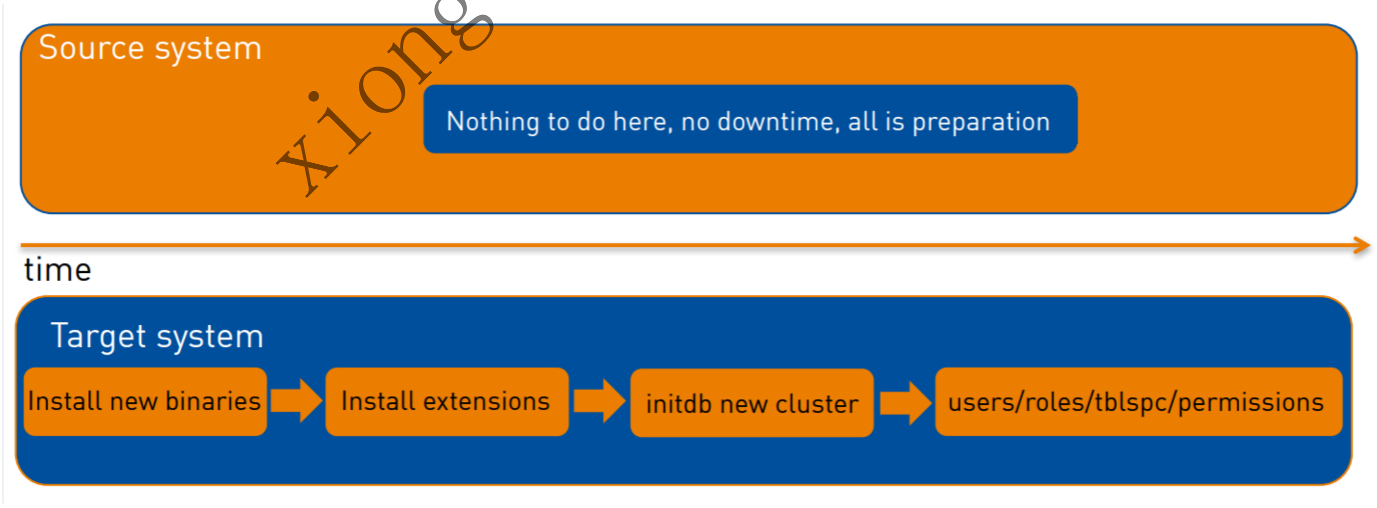
1. pg_dump
2. pg_dumpall
3. pg_dumpall + pg_dump组合拳
4. pg_upgrade
5. 大于PostgreSQL 10: Logical replication
6. 小于PostgreSQL 10: pglogical, Slony, Londiste, and Bucardo.

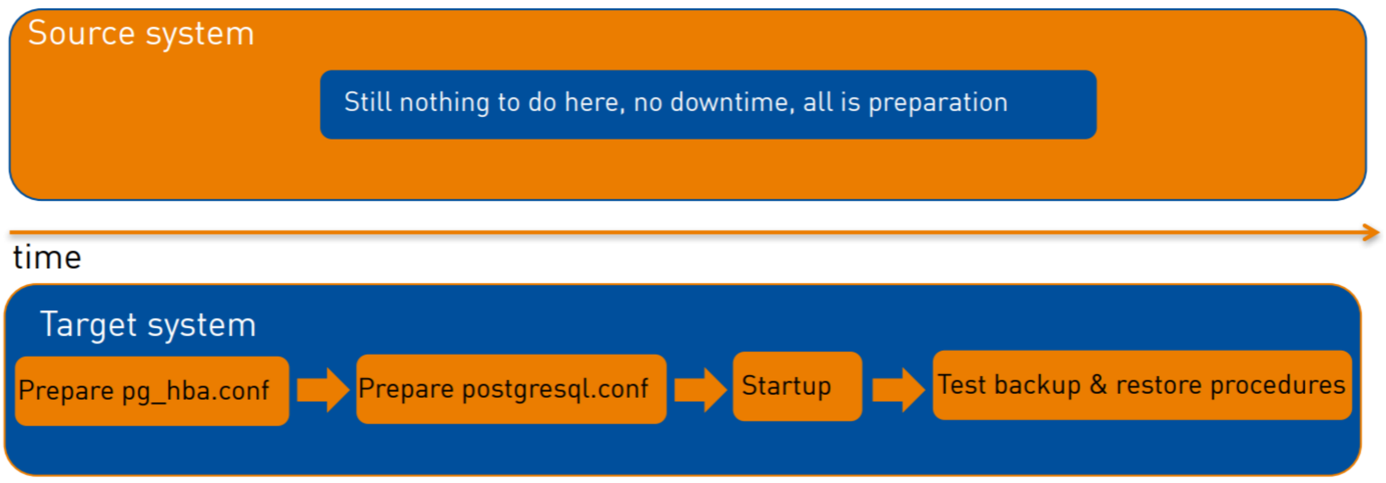
pg_dump

首先是最原始的pg_dump的方式，可以使用unix pipe，也可以使用-j 并行的方式，不过得配合pg_restore，需要额外去备份一些全局的信息如用户和表空间等。

Tooling	Task	Reloading data	Downtime needed
pg_dump	Minor release update	Not needed	Close to zero

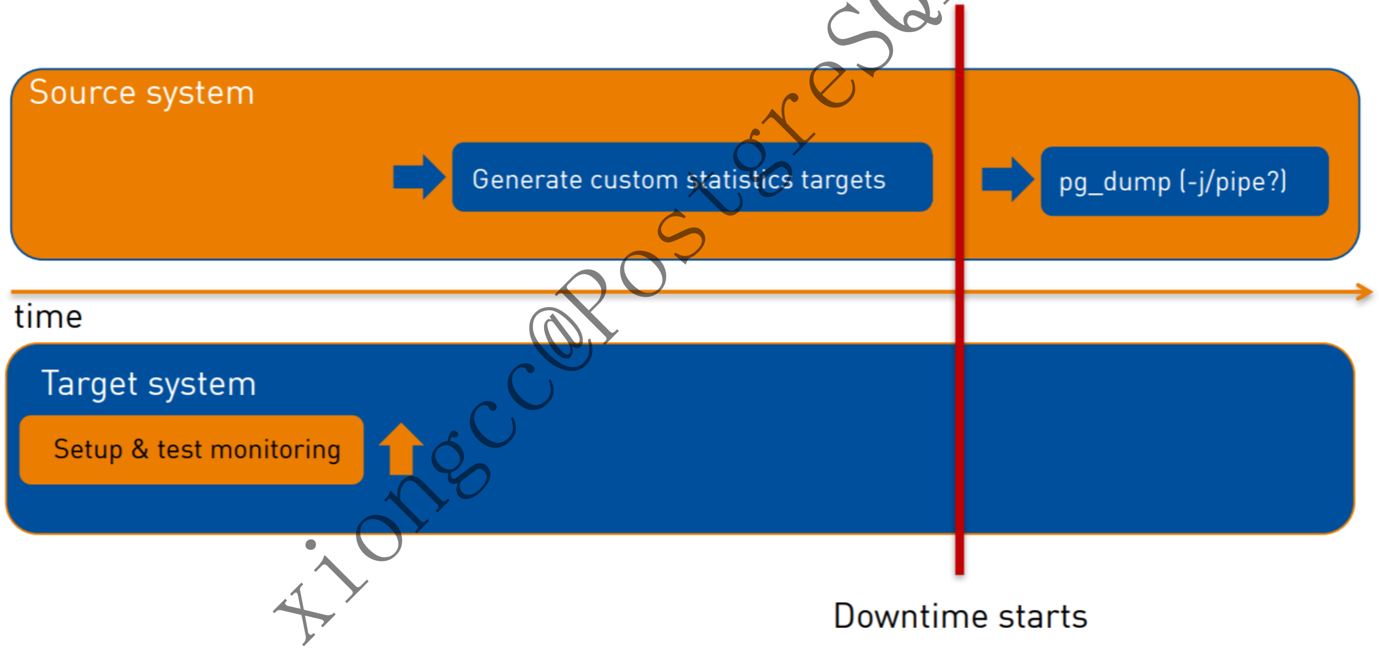


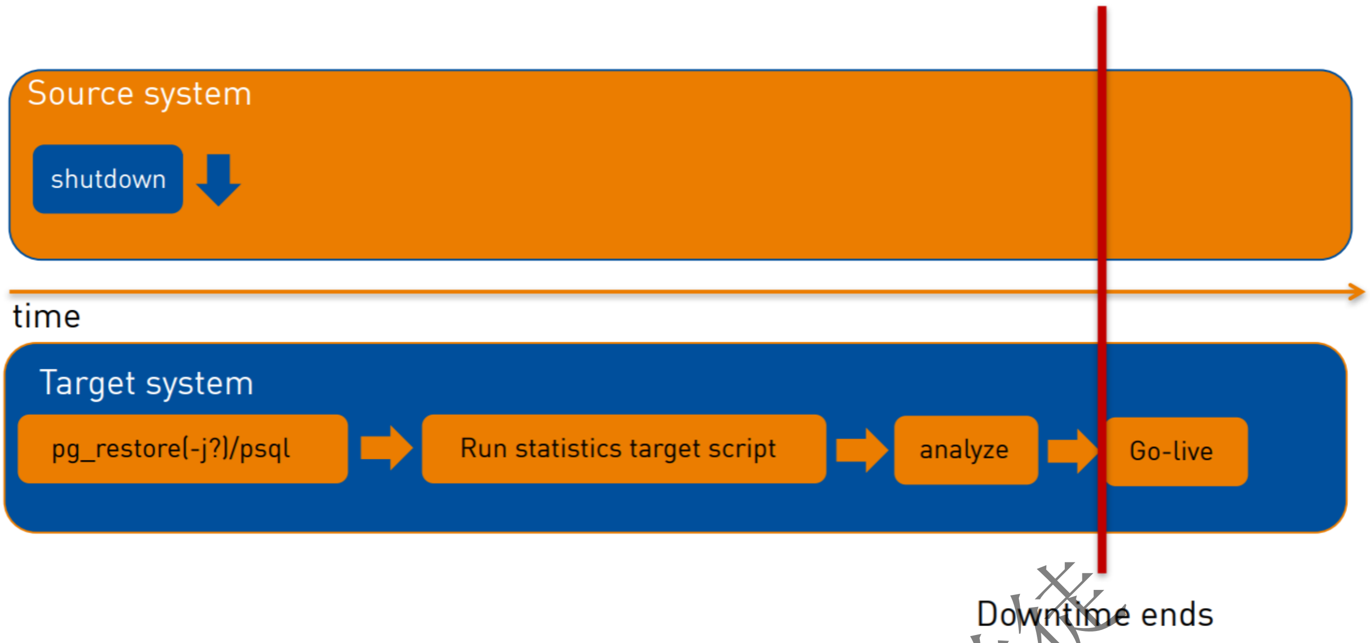



Source system
Still nothing to do here, no downtime, all is preparation
time
Target system
Prepare pg_hba.conf
Prepare postgresql.conf
Startup
Test backup & restore procedures

目标库准备工作一切就绪之后，然后使用 pg_dump并行导出，或者直接一个管道的方式，这样可以进一步减少停机的时间。

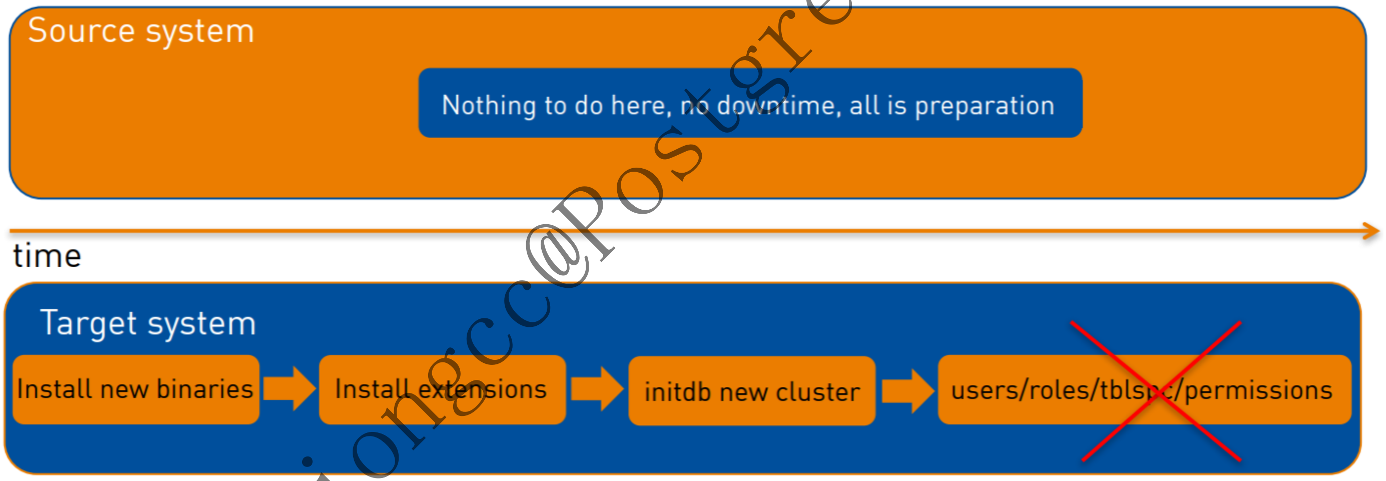
```
sudo -u postgres pg_dump -h /path/to/old/instance.sock | sudo -u postgres psql -h
/path/to/new/instance.sock
```

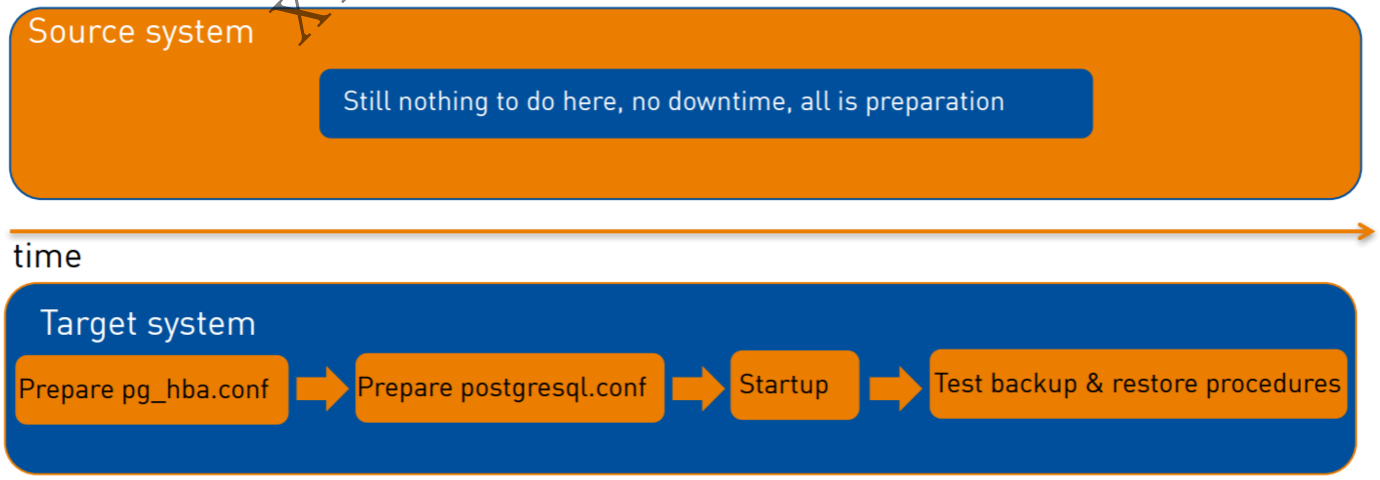


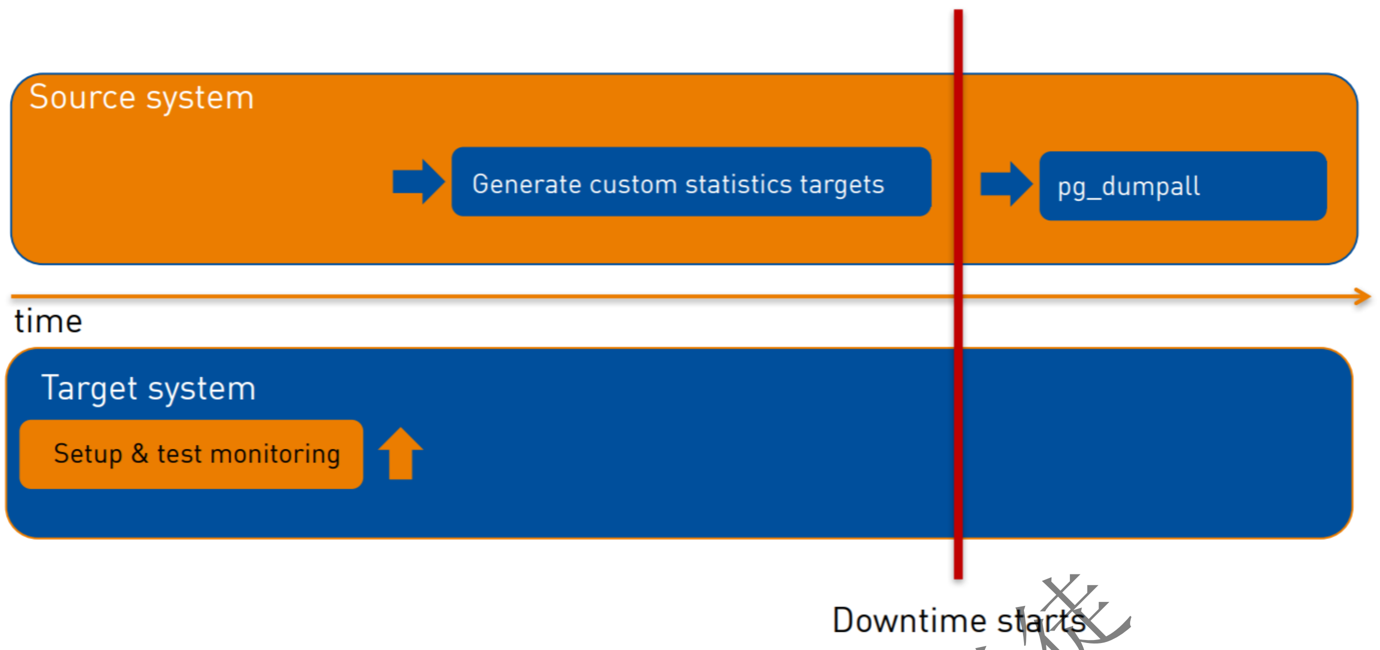


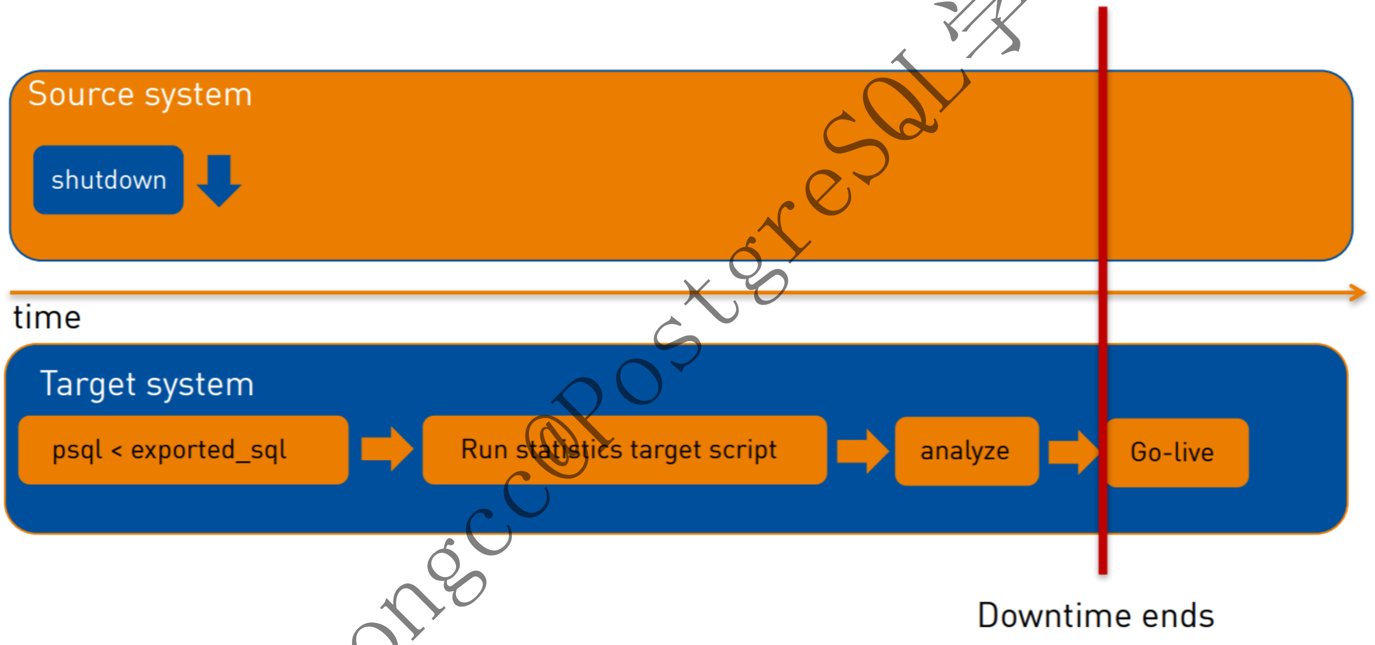
pg_dumpall

pg_dumpall 的优势在于，可以导出全局的对象，如用户、表空间、全局的一些权限信息等等，所以相较于pg_dump就不需要导出用户信息等全局信息这一步骤了，不过不支持-j并行，和pg_dump的步骤十分类似。




Source system
Still nothing to do here, no downtime, all is preparation
time
Target system
Prepare pg_hba.conf
Prepare postgresql.conf
Startup
Test backup & restore procedures

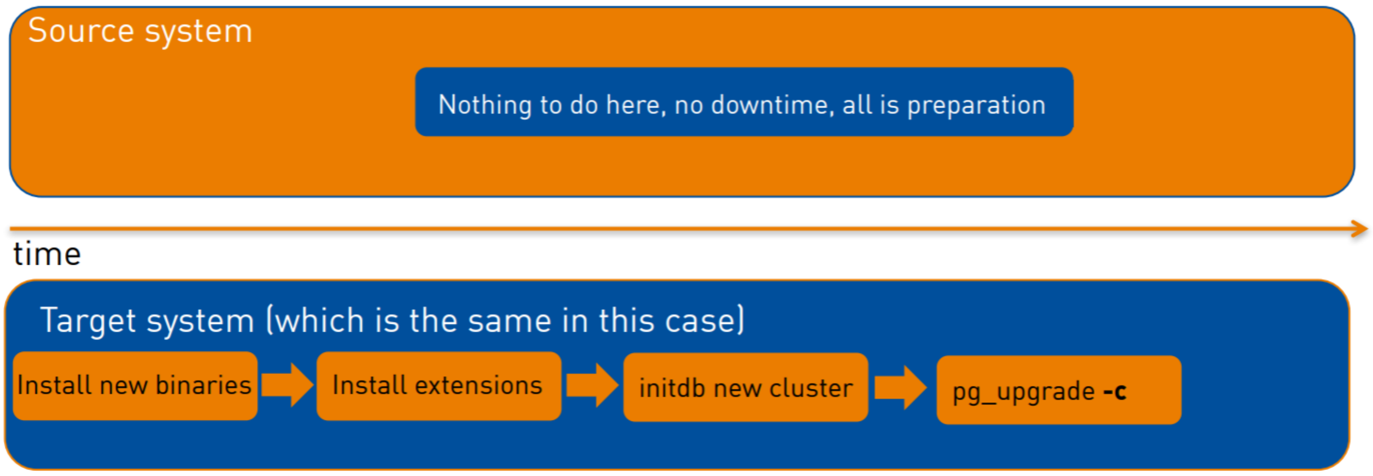




pg_upgrade

pg_upgrade是官方提供的一个升级工具，执行升级之前，需要使用 -c 或者 --check选项，进行升级前的兼容性检查， 可以发现插件、数据类型不兼容等问题。不过需要注意的是，源端和目标端的编译参数、segsize、walsegsize等等得是一样的，可以使用pg_config查看源端的编译参数

```
[postgres@xiongcc ~]$ pg_config  | egrep 'CONFIGURE|CC'
CONFIGURE =  '--prefix=/usr/pgsql-13' '--enable-debug' '--enable-cassert' '--enable-depend' 'CFLAGS=-O0 -ggdb' '--with-readline'
CC = gcc -std=gnu99
```


Source system
Nothing to do here, no downtime, all is preparation
time
Target system (which is the same in this case)
Install new binaries
Install extensions
initdb new cluster
pg_upgrade -c

兼容性检测结果，类似如下：

```
[postgres@xiongcc ~]$ pg_upgrade --old-datadir /home/postgres/12data/ --new-datadir
/home/postgres/upgrade_test/ --old-bindir=/usr/pgsql-12/bin/ --new-bindir=/usr/pgsql-
13/bin/ --check
Performing Consistency Checks on Old Live Server
------------------------------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for presence of required libraries                 ok
Checking database user is the install user                  ok
Checking for prepared transactions                          ok
Checking for new cluster tablespace directories             ok

*Clusters are compatible*
```

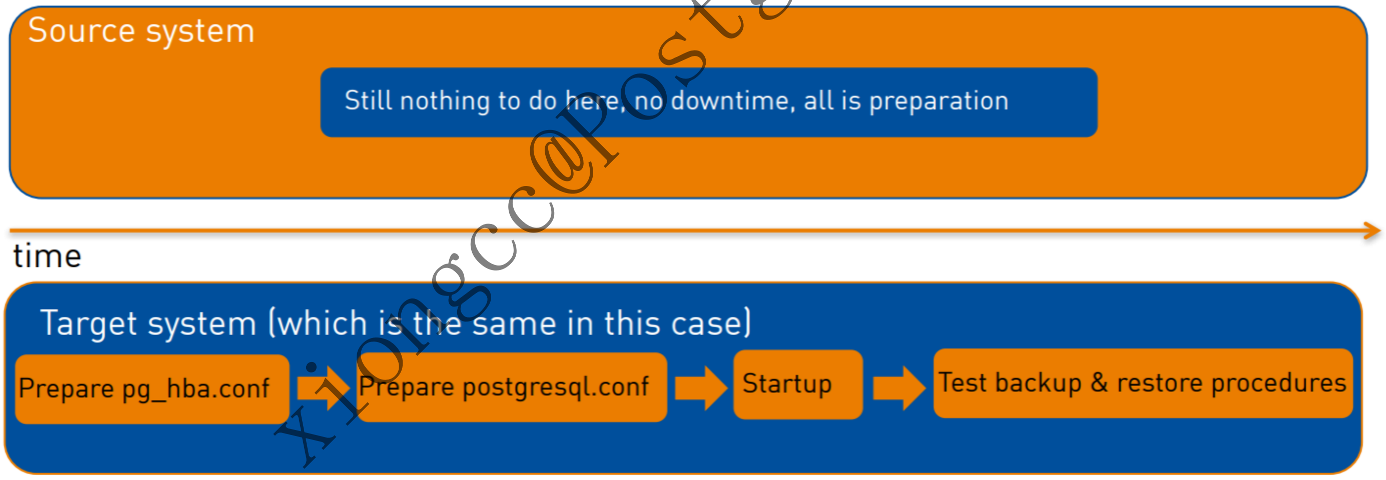
常见的兼容错误如下：

1. Upgrade Error - There seems to be a postmaster servicing the cluster，服务还在运行中，所以这也算是pg_upgrade的一个小小缺点，需要停机
2. Upgrade Error - fe_sendauth: no password supplied，这个就简单处理，密码错误，指定pg_hba.conf或者.pgpass
3. Upgrade Error - New cluster is not empty; exiting，需要保证目标库的data目录是空的，比如New cluster database "postgres" is not empty: found relation "public.test"，发现目标端里面已经建表了。
4. Upgrade Error - Failed to load library，这个是最常见的，源端有一些第三方插件，而目标端没有，这个需要自己安装好对应的.so动态库即可

```
[postgres@xiongcc ~]$ pg_upgrade --old-datadir /home/postgres/12data/ --new-datadir
/home/postgres/upgrade_test/ --old-bindir=/usr/pgsql-12/bin/ --new-bindir=/usr/pgsql-
13/bin/ --check
Performing Consistency Checks
```

```
-----------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for presence of required libraries                 fatal

Your installation references loadable libraries that are missing from the
new installation.  You can add these libraries to the new installation,
or remove the functions using them from the old installation.  A list of
problem libraries is in the file:
    loadable_libraries.txt

Failure, exiting
[postgres@xiongcc ~]$ cat loadable_libraries.txt
could not load library "pg_pathman": ERROR:  pg_pathman module must be initialized by
Postmaster. Put the following line to configuration file:
shared_preload_libraries='pg_pathman'
In database: postgres
```

老样子，先将配置文件配置好，直接 cp 源端的即可，然后就可以升级了


Source system
Still nothing to do here, no downtime, all is preparation
time
Target system (which is the same in this case)
Prepare pg_hba.conf
Prepare postgresql.conf
Startup
Test backup & restore procedures

对于pg_upgrade，提供了两种方式，一个是指定--link的方式，这样的话新版本数据库可以直接使用原有的数据库文件而不需要执行复制，通常可以在几分钟内完成升级操作，接近于 0 停机时间。不过值得注意的是，源端和目标端的编码一定要保持一致，不然就会失败。Note that pg_upgrade is only going to work in case the encodings of the old and the new database instance match. Otherwise, it will fail.

这里，还有一个隐含的concern：既然是link模式，所以前提是PostgreSQL去做链接的操作是可以成功的，不然当然也不可行，比如不能跨文件系统（即分区）进行创建。

不过link模式看似美好，实则不然，假如我以link的模式升级升上去了，我又因各种原因要回退，这里就复杂了

- If the `--check` option was used, the old cluster was unmodified; it can be restarted.
- If the `--link` option was *not* used, the old cluster was unmodified; it can be restarted.

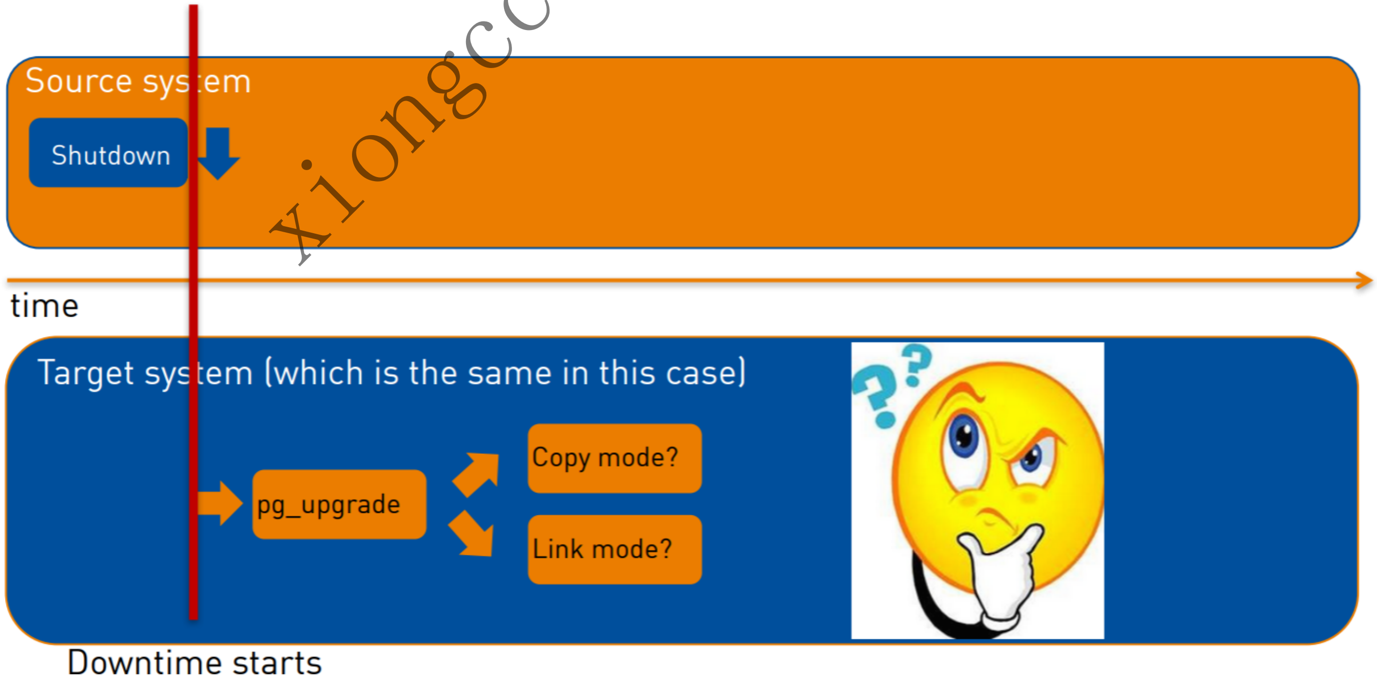
- If the `--link` option was used, the data files might be shared between the old and new cluster:
    - If `pg_upgrade` aborted before linking started, the old cluster was unmodified; it can be restarted.
    - If you did *not* start the new cluster, the old cluster was unmodified except that, when linking started, a `.old` suffix was appended to `$PGDATA/global/pg_control`. To reuse the old cluster, remove the `.old` suffix from `$PGDATA/global/pg_control`; you can then restart the old cluster.
    - If you did start the new cluster, it has written to shared files and it is unsafe to use the old cluster. The old cluster will need to be restored from backup in this case.

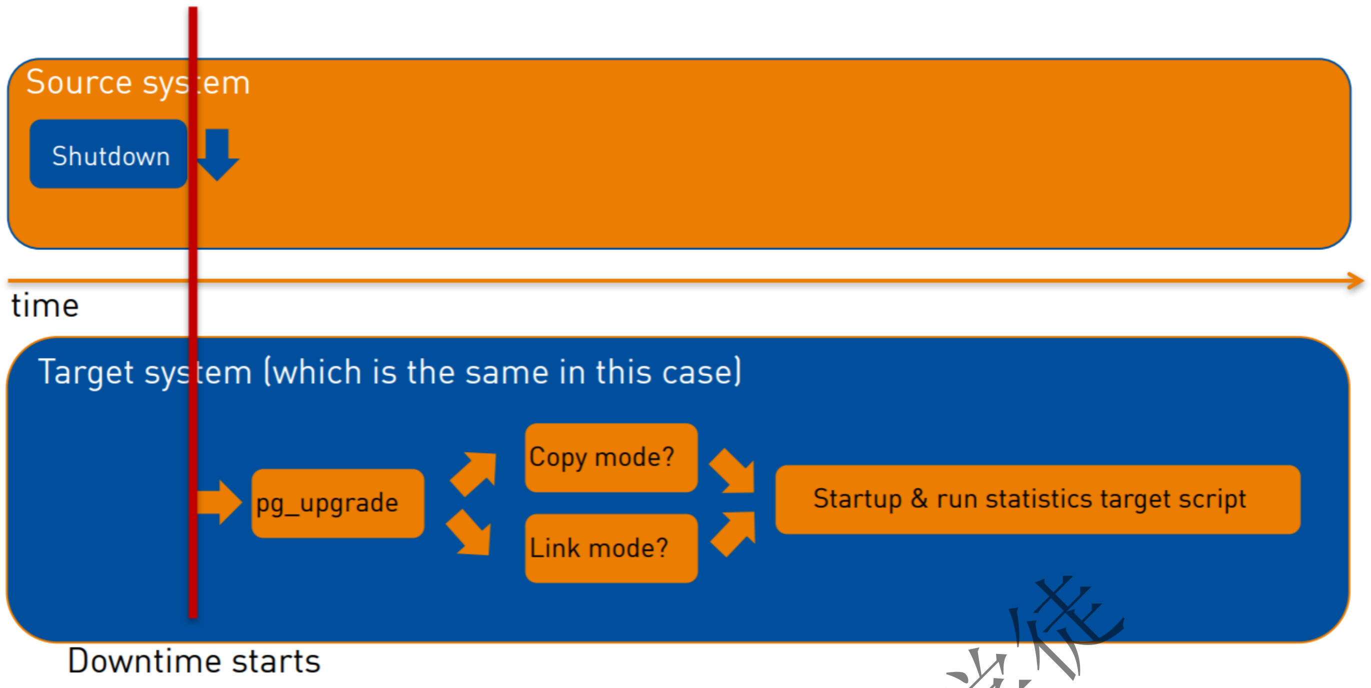
1. 如果只运行了 --check 选项命令，表示没有真正执行升级，重新启动服务即可；
2. 如果升级时没有使用 --link 选项，旧版本的数据库集群没有任何修改，重新启动服务即可；
3. 如果升级时使用了 --link 选项，数据库文件可能已经被新版本的集群使用：

- 如果 pg_upgrade 在链接操作之前终止，旧版本的数据库集群没有任何修改，重新启动服务即可；
- 如果没有启动过新版本的后台服务，旧版本的数据库集群没有修改，但是链接过程已经将 $PGDATA/global/pg_control 文件重命名为 $PGDATA/global/pg_control.old；此时需要将该文件名中的 .old 后缀去掉，然后重新启动服务即可；
- 如果已经启动了新版本的数据库集群，已经修改了数据库文件，再启动旧版本的服务可能导致数据损坏；此时需要通过备份文件还原旧版本的数据库。

另外一个模式就是不指定link了，这样就是硬生生的拷贝数据文件了，具体的速度取决于你的源端数据库大小。对于含有replica的数据库，可以使用rsync的方式进行升级，具体参照官网：https://www.postgresql.org/docs/current/pgupgrade.html

Tooling	Task	Reloading data	Downtime needed
pg_upgrade	Major release update	Copy is needed	Downtime is needed
pg_upgrade –link	Major release update	Only hardlinks	Close to zero




Source system
Shutdown
time
Target system (which is the same in this case)
pg_upgrade
Copy mode?
Link mode?
Startup & run statistics target script
Downtime starts

测试一下升级速度，大概花了10s

```
[postgres@xiongcc ~]$ pg_upgrade --old-datadir /home/postgres/12data/ --new-datadir
/home/postgres/upgrade_test/ --old-bindir=/usr/pgsql-12/bin/ --new-bindir=/usr/pgsql-
13/bin/ --link
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Creating dump of global objects                             ok
Creating dump of database schemas
                                                            ok
Checking for presence of required libraries                 ok
Checking database user is the install user                  ok
Checking for prepared transactions                          ok
Checking for new cluster tablespace directories             ok

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
------------------
Analyzing all rows in the new cluster                       ok
Freezing all rows in the new cluster                        ok
Deleting files from new pg_xact                             ok
Copying old pg_xact to new server                           ok
Setting next transaction ID and epoch for new cluster       ok
```

```
Deleting files from new pg_multixact/offsets                ok
Copying old pg_multixact/offsets to new server              ok
Deleting files from new pg_multixact/members                ok
Copying old pg_multixact/members to new server              ok
Setting next multixact ID and offset for new cluster        ok
Resetting WAL archives                                      ok
Setting frozenxid and minmxid counters in new cluster       ok
Restoring global objects in the new cluster                 ok
Restoring database schemas in the new cluster
                                                            ok
Adding ".old" suffix to old global/pg_control               ok

If you want to start the old cluster, you will need to remove
the ".old" suffix from /home/postgres/12data/global/pg_control.old.
Because "link" mode was used, the old cluster cannot be safely
started once the new cluster has been started.

Linking user relation files
                                                            ok
Setting next OID for new cluster                            ok
Sync data directory to disk                                 ok
Creating script to analyze new cluster                      ok
Creating script to delete old cluster                       ok

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade so,
once you start the new server, consider running:
    ./analyze_new_cluster.sh

Running this script will delete the old cluster's data files:
    ./delete_old_cluster.sh
```

pg_upgrade 升级之后，不会生成新版本数据库的统计信息，但是会创建一个脚本文件，执行该文件：./analyze_new_cluster.sh，可以看到，最开始会收集一个最基本的统计信息，确保数据库可用，然后再收集一次更加全面的统计信息，分阶段的，另外假如修改了default_statistics_target，可以直接使用 **"/usr/local/bin/vacuumdb" --all --analyze-only**，因为升级过程目前不会复制数据分布的任何统计数据，比如像直方图、最常见的值及其频率之类的东西，所以这也是"vacuumdb –analyze-only –analyze-in-stage"的重要性，不然复杂查询的性能会受到相当大的影响

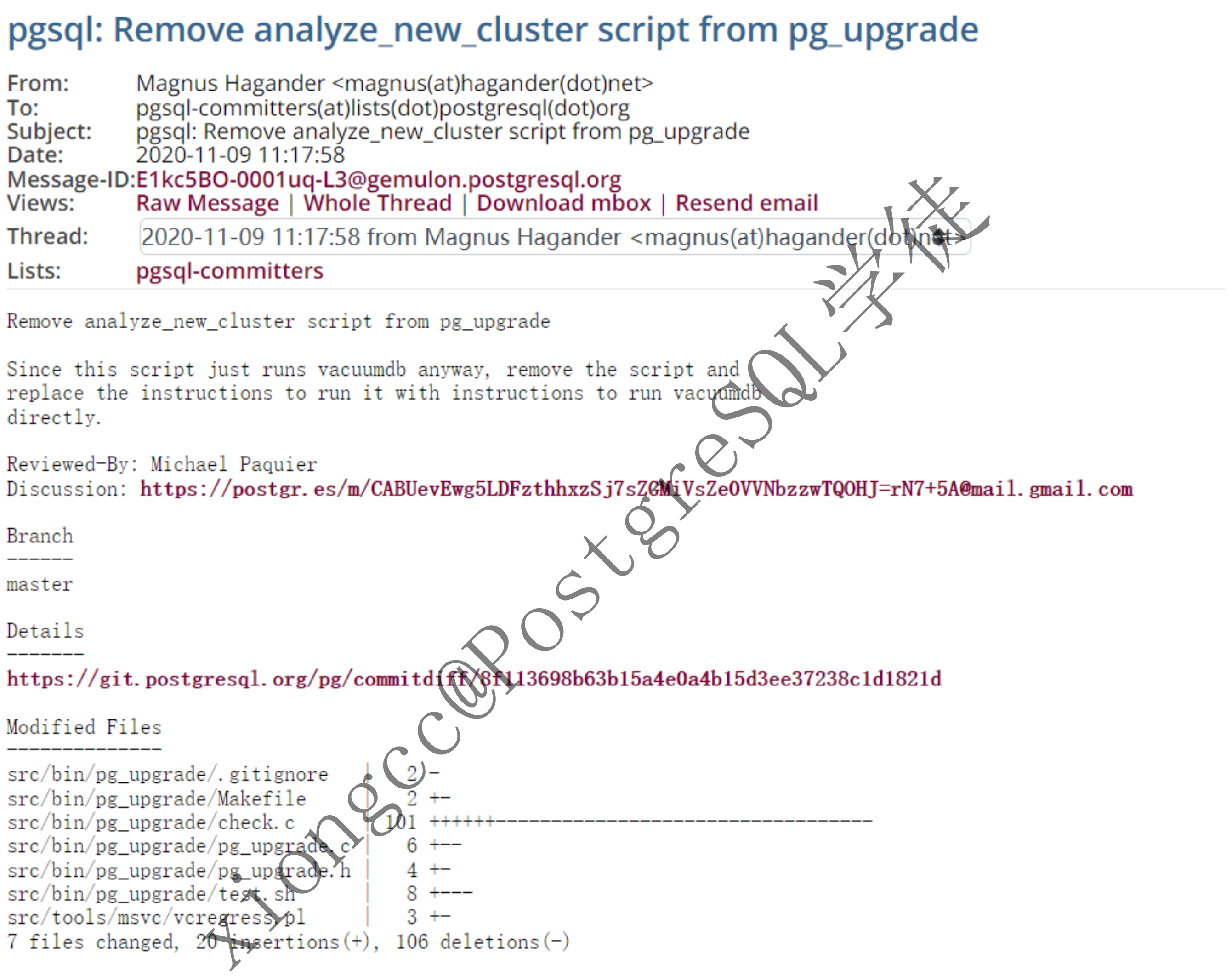
```
[postgres@xiongcc ~]$ cat analyze_new_cluster.sh
#!/bin/sh

echo 'This script will generate minimal optimizer statistics rapidly'
echo 'so your system is usable, and then gather statistics twice more'
echo 'with increasing accuracy.  When it is done, your system will'
echo 'have the default level of optimizer statistics.'
echo
```

```
echo 'If you have used ALTER TABLE to modify the statistics target for'
echo 'any tables, you might want to remove them and restore them after'
echo 'running this script because they will delay fast statistics generation.'
echo

echo 'If you would like default statistics as quickly as possible, cancel'
echo 'this script and run:'
echo '    "/usr/pgsql-13/bin/vacuumdb" --all --analyze-only'
echo

"/usr/pgsql-13/bin/vacuumdb" --all --analyze-in-stages
echo

echo 'Done'

[postgres@xiongcc ~]$ cat delete_old_cluster.sh
#!/bin/sh

rm -rf '/home/postgres/12data'
```

```
[postgres@xiongcc ~]$ ./analyze_new_cluster.sh
This script will generate minimal optimizer statistics rapidly
so your system is usable, and then gather statistics twice more
with increasing accuracy.  When it is done, your system will
have the default level of optimizer statistics.

If you have used ALTER TABLE to modify the statistics target for
any tables, you might want to remove them and restore them after
running this script because they will delay fast statistics generation.

If you would like default statistics as quickly as possible, cancel
this script and run:
    "/usr/pgsql-13/bin/vacuumdb" --all --analyze-only

vacuumdb: processing database "mydb": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "postgres": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "template1": Generating minimal optimizer statistics (1
target)
vacuumdb: processing database "mydb": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "postgres": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "template1": Generating medium optimizer statistics (10
targets)
vacuumdb: processing database "mydb": Generating default (full) optimizer statistics
```

```
vacuumdb: processing database "postgres": Generating default (full) optimizer
statistics
vacuumdb: processing database "template1": Generating default (full) optimizer
statistics

Done
```

在14以后，移除了analyze_new_cluster.sh



pgsql: Remove analyze_new_cluster script from pg_upgrade

From: Magnus Hagander <magnus(at)hagander(dot)net>
To: pgsql-committers(at)lists(dot)postgresql(dot)org
Subject: pgsql: Remove analyze_new_cluster script from pg_upgrade
Date: 2020-11-09 11:17:58
Message-ID: E1kc5BO-0001uq-L3@gemulon.postgresql.org
Views: Raw Message | Whole Thread | Download mbox | Resend email
Thread: 2020-11-09 11:17:58 from Magnus Hagander <magnus(at)hagander(dot)net>
Lists: pgsql-committers

```
Remove analyze_new_cluster script from pg_upgrade

Since this script just runs vacuumdb anyway, remove the script and
replace the instructions to run it with instructions to run vacuumdb
directly.

Reviewed-By: Michael Paquier
Discussion: https://postgr.es/m/CABUevEwg5LDFzthhxzSj7sZGMiVsZe0VVNbzzwTQOHJ=rN7+5A@mail.gmail.com

Branch
------
master

Details
-------
https://git.postgresql.org/pg/commitdiff/8f113698b63b15a4e0a4b15d3ee37238c1d1821d

Modified Files
--------------
src/bin/pg_upgrade/.gitignore    |   2 -
src/bin/pg_upgrade/Makefile      |   2 +-
src/bin/pg_upgrade/check.c       | 101 ++++++-----------------------------------
src/bin/pg_upgrade/pg_upgrade.c  |   6 +--
src/bin/pg_upgrade/pg_upgrade.h  |   4 +-
src/bin/pg_upgrade/test.sh       |   8 +---
src/tools/msvc/vcregress.pl      |   3 +-
7 files changed, 20 insertions(+), 106 deletions(-)
```

可以看到1 target、10 targets，分阶段来做的，可以使用SQL检测修改了target的表：

```
postgres=# alter table pgbench_accounts alter COLUMN aid set statistics 2048;
ALTER TABLE
postgres=# WITH tabs AS (
    SELECT
        tablename,
        schemaname
    FROM
        pg_tables
    WHERE
        schemaname NOT IN ('information_schema', 'pg_catalog'))
SELECT
```

```
    'alter table ' || b.schemaname || '.' || b.tablename || ' alter column ' ||
a.attname || ' set
statistics '||a.attstattarget||';'
FROM
    pg_attribute a,
    tabs b
WHERE
    attrelid::regclass::varchar = b.tablename
    AND attstattarget > 0;
                         ?column?
----------------------------------------------------------
 alter table public.pgbench_accounts alter column aid set+
 statistics 2048;
(1 row)

postgres=# WITH tabs AS (
    SELECT
        tablename
    FROM
        pg_tables
    WHERE
        schemaname NOT IN ('information_schema', 'pg_catalog'))
SELECT
    attrelid::regclass,
    attname,
    attstattarget
FROM
    pg_attribute a,
    tabs b
WHERE
    attrelid::regclass::varchar = b.tablename
    AND attstattarget > 0
ORDER BY
    1,
    2,
    3;
     attrelid     | attname | attstattarget
------------------+---------+---------------
 pgbench_accounts | aid     |          2048
(1 row)
```

可以加上-v verbose打印详细信息，可以方便我们了解pg_upgrade的原理

```
[postgres@xiongcc ~]$ pg_upgrade --old-datadir /home/postgres/9data/ --new-datadir
/home/postgres/xcc_up/ --old-bindir=/usr/pgsql-9.6/bin/ --new-bindir=/usr/pgsql-13/bin/
--link -v
Running in verbose mode
Performing Consistency Checks
-----------------------------
```

```
Checking cluster versions                                   ok
Current pg_control values:

pg_control version number:            960
Catalog version number:               201608131
Database system identifier:           6996978409635986048
Latest checkpoint's TimeLineID:       1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:1760
Latest checkpoint's NextOID:          16387
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        1750
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  0
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
Latest checkpoint's newestCommitTsXid:0
Maximum data alignment:               8
Database block size:                  8192
Blocks per segment of large relation: 131072
WAL block size:                       8192
Bytes per WAL segment:                16777216
Maximum length of identifiers:        64
Maximum columns in an index:          32
Maximum size of a TOAST chunk:        1996
Size of a large-object chunk:         2048
Date/time type storage:               64-bit integers
Float4 argument passing:              by value
Float8 argument passing:              by value
Data page checksum version:           0


Values to be changed:

First log segment after reset:        000000010000000000000004
Current pg_control values:

pg_control version number:            1300
Catalog version number:               202007201
Database system identifier:           6996978613898283695
Latest checkpoint's TimeLineID:       1
Latest checkpoint's full_page_writes: off
Latest checkpoint's NextXID:          0:485
Latest checkpoint's NextOID:          13579
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        478
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  0
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
Latest checkpoint's newestCommitTsXid:0
Maximum data alignment:               8
Database block size:                  8192
Blocks per segment of large relation: 131072
WAL block size:                       8192
Bytes per WAL segment:                16777216
Maximum length of identifiers:        64
Maximum columns in an index:          32
Maximum size of a TOAST chunk:        1996
Size of a large-object chunk:         2048
Date/time type storage:               64-bit integers
Float8 argument passing:              by value
Data page checksum version:           0


Values to be changed:

First log segment after reset:        000000010000000000000002
"/usr/pgsql-9.6/bin/pg_ctl" -w -l "pg_upgrade_server.log" -D "/home/postgres/9data" -o
"-p 50432 -b  -c listen_addresses='' -c unix_socket_permissions=0700 -c
unix_socket_directories='/home/postgres'" start >> "pg_upgrade_server.log" 2>&1
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT d.oid, d.datname, d.encoding, d.datcollate, d.datctype,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM pg_catalog.pg_database d
 LEFT OUTER JOIN pg_catalog.pg_tablespace t  ON d.dattablespace = t.oid WHERE
d.datallowconn = true ORDER BY 2
executing: SELECT pg_catalog.set_config('search_path', '', false);
```

```
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n        ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                          'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR      (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c       ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n        ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                          'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR      (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c       ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;

source databases:
Database: postgres
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:
relname: public.test: reloid: 16384 reltblspace:



Database: template1
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:
```

```
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT pg_catalog.pg_tablespace_location(oid) AS spclocation FROM
pg_catalog.pg_tablespace WHERE  spcname != 'pg_default' ANDspcname != 'pg_global'
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT DISTINCT probin FROM pg_catalog.pg_proc WHERE prolang = 13 AND probin
IS NOT NULL AND oid >= 16384;
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT DISTINCT probin FROM pg_catalog.pg_proc WHERE prolang = 13 AND probin
IS NOT NULL AND oid >= 16384;
executing: SELECT pg_catalog.set_config('search_path', '', false);
Checking database user is the install user                 executing: SELECT rolsuper,
oid FROM pg_catalog.pg_roles WHERE rolname = current_user AND rolname !~ '^pg_'
executing: SELECT COUNT(*) FROM pg_catalog.pg_roles WHERE rolname !~ '^pg_'
ok
Checking database connection settings                      executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: SELECT datname, datallowconn FROM  pg_catalog.pg_database
ok
executing: SELECT pg_catalog.set_config('search_path', '', false);
Checking for prepared transactions                         executing: SELECT * FROM
pg_catalog.pg_prepared_xacts
ok
Checking for reg* data types in user tables                executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname, a.attname FROM  pg_catalog.pg_class c,
pg_catalog.pg_namespace n,    pg_catalog.pg_attribute a,    pg_catalog.pg_type t WHERE
c.oid = a.attrelid AND    NOT a.attisdropped AND        a.atttypid = t.oid AND
t.typnamespace =             (SELECT oid FROM pg_namespace             WHERE nspname =
'pg_catalog') AND   t.typname IN (             'regcollation',            'regconfig',
         'regdictionary',          'regnamespace',            'regoper',
 'regoperator',            'regproc',            'regprocedure'       ) AND
c.relnamespace = n.oid AND    n.nspname NOT IN ('pg_catalog', 'information_schema')
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname, a.attname FROM  pg_catalog.pg_class c,
pg_catalog.pg_namespace n,    pg_catalog.pg_attribute a,    pg_catalog.pg_type t WHERE
c.oid = a.attrelid AND    NOT a.attisdropped AND        a.atttypid = t.oid AND
t.typnamespace =             (SELECT oid FROM pg_namespace             WHERE nspname =
'pg_catalog') AND   t.typname IN (             'regcollation',            'regconfig',
         'regdictionary',          'regnamespace',            'regoper',
 'regoperator',            'regproc',            'regprocedure'       ) AND
c.relnamespace = n.oid AND    n.nspname NOT IN ('pg_catalog', 'information_schema')
ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for tables WITH OIDS                              executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname FROM pg_catalog.pg_class c,
pg_catalog.pg_namespace n WHERE c.relnamespace = n.oid AND     c.relhasoids AND
n.nspname NOT IN ('pg_catalog')
```

```
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname FROM pg_catalog.pg_class c,
pg_catalog.pg_namespace n WHERE c.relnamespace = n.oid AND     c.relhasoids AND
n.nspname NOT IN ('pg_catalog')
ok
Checking for invalid "sql_identifier" user columns          executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: WITH RECURSIVE oids AS (    SELECT
'information_schema.sql_identifier'::pg_catalog.regtype AS oid   UNION ALL   SELECT *
FROM (     WITH x AS (SELECT oid FROM oids)        SELECT t.oid FROM pg_catalog.pg_type
t, x WHERE typbasetype = x.oid AND typtype = 'd'       UNION ALL      SELECT t.oid FROM
pg_catalog.pg_type t, x WHERE typelem = x.oid AND typtype = 'b'        UNION ALL  SELECT
t.oid FROM pg_catalog.pg_type t, pg_catalog.pg_class c, pg_catalog.pg_attribute a, x
WHERE t.typtype = 'c' AND          t.oid = c.reltype AND           c.oid = a.attrelid AND
 NOT a.attisdropped AND          a.atttypid = x.oid       UNION ALL       SELECT t.oid
FROM pg_catalog.pg_type t, pg_catalog.pg_range r, x     WHERE t.typtype = 'r' AND
r.rngtypid = t.oid AND r.rngsubtype = x.oid   ) foo ) SELECT n.nspname, c.relname,
a.attname FROM pg_catalog.pg_class c,    pg_catalog.pg_namespace n,
pg_catalog.pg_attribute a WHERE c.oid = a.attrelid AND    NOT a.attisdropped AND
a.atttypid IN (SELECT oid FROM oids) AND     c.relkind IN ('r', 'm', 'i') AND
c.relnamespace = n.oid AND    n.nspname !~ '^pg_temp_' AND    n.nspname !~
'^pg_toast_temp_' AND    n.nspname NOT IN ('pg_catalog', 'information_schema')
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH RECURSIVE oids AS (    SELECT
'information_schema.sql_identifier'::pg_catalog.regtype AS oid   UNION ALL   SELECT *
FROM (     WITH x AS (SELECT oid FROM oids)        SELECT t.oid FROM pg_catalog.pg_type
t, x WHERE typbasetype = x.oid AND typtype = 'd'       UNION ALL      SELECT t.oid FROM
pg_catalog.pg_type t, x WHERE typelem = x.oid AND typtype = 'b'        UNION ALL  SELECT
t.oid FROM pg_catalog.pg_type t, pg_catalog.pg_class c, pg_catalog.pg_attribute a, x
WHERE t.typtype = 'c' AND          t.oid = c.reltype AND           c.oid = a.attrelid AND
 NOT a.attisdropped AND          a.atttypid = x.oid       UNION ALL       SELECT t.oid
FROM pg_catalog.pg_type t, pg_catalog.pg_range r, x     WHERE t.typtype = 'r' AND
r.rngtypid = t.oid AND r.rngsubtype = x.oid   ) foo ) SELECT n.nspname, c.relname,
a.attname FROM pg_catalog.pg_class c,    pg_catalog.pg_namespace n,
pg_catalog.pg_attribute a WHERE c.oid = a.attrelid AND    NOT a.attisdropped AND
a.atttypid IN (SELECT oid FROM oids) AND     c.relkind IN ('r', 'm', 'i') AND
c.relnamespace = n.oid AND    n.nspname !~ '^pg_temp_' AND    n.nspname !~
'^pg_toast_temp_' AND    n.nspname NOT IN ('pg_catalog', 'information_schema')
ok
Checking for invalid "unknown" user columns                 executing: SELECT
pg_catalog.set_config('search_path', '', false);
```

```
executing: WITH RECURSIVE oids AS (   SELECT 'pg_catalog.unknown'::pg_catalog.regtype
AS oid  UNION ALL   SELECT * FROM (     WITH x AS (SELECT oid FROM oids)      SELECT
t.oid FROM pg_catalog.pg_type t, x WHERE typbasetype = x.oid AND typtype = 'd'
UNION ALL      SELECT t.oid FROM pg_catalog.pg_type t, x WHERE typelem = x.oid AND
typtype = 'b'       UNION ALL  SELECT t.oid FROM pg_catalog.pg_type t,
pg_catalog.pg_class c, pg_catalog.pg_attribute a, x      WHERE t.typtype = 'c' AND
  t.oid = c.reltype AND          c.oid = a.attrelid AND          NOT a.attisdropped AND
    a.atttypid = x.oid       UNION ALL       SELECT t.oid FROM pg_catalog.pg_type t,
pg_catalog.pg_range r, x     WHERE t.typtype = 'r' AND r.rngtypid = t.oid AND
r.rngsubtype = x.oid   ) foo ) SELECT n.nspname, c.relname, a.attname FROM
pg_catalog.pg_class c,    pg_catalog.pg_namespace n,    pg_catalog.pg_attribute a WHERE
c.oid = a.attrelid AND    NOT a.attisdropped AND     a.atttypid IN (SELECT oid FROM
oids) AND     c.relkind IN ('r', 'm', 'i') AND    c.relnamespace = n.oid AND
n.nspname !~ '^pg_temp_' AND    n.nspname !~ '^pg_toast_temp_' AND    n.nspname NOT IN
('pg_catalog', 'information_schema')
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH RECURSIVE oids AS (   SELECT 'pg_catalog.unknown'::pg_catalog.regtype
AS oid  UNION ALL   SELECT * FROM (     WITH x AS (SELECT oid FROM oids)      SELECT
t.oid FROM pg_catalog.pg_type t, x WHERE typbasetype = x.oid AND typtype = 'd'
UNION ALL      SELECT t.oid FROM pg_catalog.pg_type t, x WHERE typelem = x.oid AND
typtype = 'b'       UNION ALL  SELECT t.oid FROM pg_catalog.pg_type t,
pg_catalog.pg_class c, pg_catalog.pg_attribute a, x      WHERE t.typtype = 'c' AND
  t.oid = c.reltype AND          c.oid = a.attrelid AND          NOT a.attisdropped AND
    a.atttypid = x.oid       UNION ALL       SELECT t.oid FROM pg_catalog.pg_type t,
pg_catalog.pg_range r, x     WHERE t.typtype = 'r' AND r.rngtypid = t.oid AND
r.rngsubtype = x.oid   ) foo ) SELECT n.nspname, c.relname, a.attname FROM
pg_catalog.pg_class c,    pg_catalog.pg_namespace n,    pg_catalog.pg_attribute a WHERE
c.oid = a.attrelid AND    NOT a.attisdropped AND     a.atttypid IN (SELECT oid FROM
oids) AND     c.relkind IN ('r', 'm', 'i') AND    c.relnamespace = n.oid AND
n.nspname !~ '^pg_temp_' AND    n.nspname !~ '^pg_toast_temp_' AND    n.nspname NOT IN
('pg_catalog', 'information_schema')
ok
Creating dump of global objects                             "/usr/pgsql-
13/bin/pg_dumpall" --host /home/postgres --port 50432 --username postgres --globals-
only --quote-all-identifiers --binary-upgrade --verbose -f pg_upgrade_dump_globals.sql
>> "pg_upgrade_utility.log" 2>&1
ok
Creating dump of database schemas
"/usr/pgsql-13/bin/pg_dump" --host /home/postgres --port 50432 --username postgres --
schema-only --quote-all-identifiers --binary-upgrade --format=custom --verbose --
file="pg_upgrade_dump_13323.custom" 'dbname=postgres' >> "pg_upgrade_dump_13323.log"
2>&1
"/usr/pgsql-13/bin/pg_dump" --host /home/postgres --port 50432 --username postgres --
schema-only --quote-all-identifiers --binary-upgrade --format=custom --verbose --
file="pg_upgrade_dump_1.custom" 'dbname=template1' >> "pg_upgrade_dump_1.log" 2>&1
                                                                   ok
"/usr/pgsql-9.6/bin/pg_ctl" -w -D "/home/postgres/9data" -o "" -m smart stop >>
"pg_upgrade_server.log" 2>&1
```

```
"/usr/pgsql-13/bin/pg_ctl" -w -l "pg_upgrade_server.log" -D "/home/postgres/xcc_up" -o
"-p 50432 -b -c synchronous_commit=off -c fsync=off -c full_page_writes=off -c
vacuum_defer_cleanup_age=0  -c listen_addresses='' -c unix_socket_permissions=0700 -c
unix_socket_directories='/home/postgres'" start >> "pg_upgrade_server.log" 2>&1
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT d.oid, d.datname, d.encoding, d.datcollate, d.datctype,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM pg_catalog.pg_database d
 LEFT OUTER JOIN pg_catalog.pg_tablespace t  ON d.dattablespace = t.oid WHERE
d.datallowconn = true ORDER BY 2
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n          ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                         'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR      (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c       ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n          ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                         'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR      (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c       ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;

target databases:
```

```
Database: postgres
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:


Database: template1
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:


executing: SELECT pg_catalog.set_config('search_path', '', false);
Checking for presence of required libraries                 ok
executing: SELECT pg_catalog.set_config('search_path', '', false);
Checking database user is the install user                  executing: SELECT rolsuper,
oid FROM pg_catalog.pg_roles WHERE rolname = current_user AND rolname !~ '^pg_'
executing: SELECT COUNT(*) FROM pg_catalog.pg_roles WHERE rolname !~ '^pg_'
ok
executing: SELECT pg_catalog.set_config('search_path', '', false);
Checking for prepared transactions                          executing: SELECT * FROM
pg_catalog.pg_prepared_xacts
ok
Checking for new cluster tablespace directories             ok

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
------------------
Analyzing all rows in the new cluster                       "/usr/pgsql-
13/bin/vacuumdb" --host /home/postgres --port 50432 --username postgres --all --analyze
--verbose >> "pg_upgrade_utility.log" 2>&1
ok
Freezing all rows in the new cluster                        "/usr/pgsql-
13/bin/vacuumdb" --host /home/postgres --port 50432 --username postgres --all --freeze
--verbose >> "pg_upgrade_utility.log" 2>&1
ok
"/usr/pgsql-13/bin/pg_ctl" -w -D "/home/postgres/xcc_up" -o "" -m smart stop >>
"pg_upgrade_server.log" 2>&1
Deleting files from new pg_xact                             ok
Copying old pg_clog to new server                           cp -Rf
"/home/postgres/9data/pg_clog" "/home/postgres/xcc_up/pg_xact" >>
"pg_upgrade_utility.log" 2>&1
ok
Setting next transaction ID and epoch for new cluster       "/usr/pgsql-
13/bin/pg_resetwal" -f -x 1760 "/home/postgres/xcc_up" >> "pg_upgrade_utility.log" 2>&1
"/usr/pgsql-13/bin/pg_resetwal" -f -e 0 "/home/postgres/xcc_up" >>
"pg_upgrade_utility.log" 2>&1
"/usr/pgsql-13/bin/pg_resetwal" -f -c 1760,1760 "/home/postgres/xcc_up" >>
"pg_upgrade_utility.log" 2>&1
```

```
ok
Deleting files from new pg_multixact/offsets                ok
Copying old pg_multixact/offsets to new server              cp -Rf
"/home/postgres/9data/pg_multixact/offsets"
"/home/postgres/xcc_up/pg_multixact/offsets" >> "pg_upgrade_utility.log" 2>&1
ok
Deleting files from new pg_multixact/members                ok
Copying old pg_multixact/members to new server              cp -Rf
"/home/postgres/9data/pg_multixact/members"
"/home/postgres/xcc_up/pg_multixact/members" >> "pg_upgrade_utility.log" 2>&1
ok
Setting next multixact ID and offset for new cluster        "/usr/pgsql-
13/bin/pg_resetwal" -O 0 -m 1,1 "/home/postgres/xcc_up" >> "pg_upgrade_utility.log"
2>&1
ok
Resetting WAL archives                                      "/usr/pgsql-
13/bin/pg_resetwal" -l 000000010000000000000004 "/home/postgres/xcc_up" >>
"pg_upgrade_utility.log" 2>&1
ok
"/usr/pgsql-13/bin/pg_ctl" -w -l "pg_upgrade_server.log" -D "/home/postgres/xcc_up" -o
"-p 50432 -b -c synchronous_commit=off -c fsync=off -c full_page_writes=off -c
vacuum_defer_cleanup_age=0  -c listen_addresses='' -c unix_socket_permissions=0700 -c
unix_socket_directories='/home/postgres'" start >> "pg_upgrade_server.log" 2>&1
Setting frozenxid and minmxid counters in new cluster       executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: UPDATE pg_catalog.pg_database SET  datfrozenxid = '1760'
executing: UPDATE pg_catalog.pg_database SET  datminmxid = '1'
executing: SELECT datname, datallowconn FROM  pg_catalog.pg_database
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: UPDATE pg_catalog.pg_class SET relfrozenxid = '1760' WHERE relkind IN ('r',
'm', 't')
executing: UPDATE pg_catalog.pg_class SET relminmxid = '1' WHERE  relkind IN ('r', 'm',
't')
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: UPDATE pg_catalog.pg_class SET relfrozenxid = '1760' WHERE relkind IN ('r',
'm', 't')
executing: UPDATE pg_catalog.pg_class SET relminmxid = '1' WHERE  relkind IN ('r', 'm',
't')
executing: ALTER DATABASE "template0" ALLOW_CONNECTIONS = true
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: UPDATE pg_catalog.pg_class SET relfrozenxid = '1760' WHERE relkind IN ('r',
'm', 't')
executing: UPDATE pg_catalog.pg_class SET relminmxid = '1' WHERE  relkind IN ('r', 'm',
't')
executing: ALTER DATABASE "template0" ALLOW_CONNECTIONS = false
ok
```

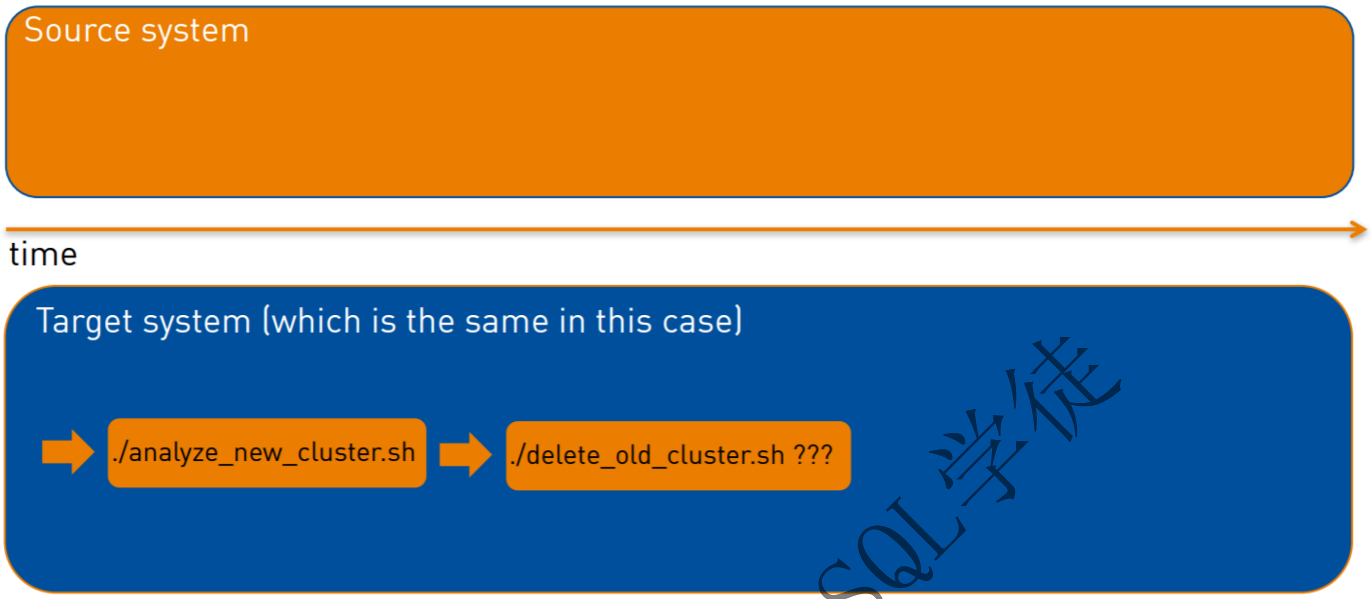
```
Restoring global objects in the new cluster                 "/usr/pgsql-13/bin/psql" --
echo-queries --set ON_ERROR_STOP=on --no-psqlrc --dbname=template1 --host
/home/postgres --port 50432 --username postgres -f "pg_upgrade_dump_globals.sql" >>
"pg_upgrade_utility.log" 2>&1
ok
Restoring database schemas in the new cluster
"/usr/pgsql-13/bin/pg_restore" --host /home/postgres --port 50432 --username postgres -
-clean --create --exit-on-error --verbose --dbname postgres "pg_upgrade_dump_1.custom"
>> "pg_upgrade_dump_1.log" 2>&1
"/usr/pgsql-13/bin/pg_restore" --host /home/postgres --port 50432 --username postgres -
-clean --create --exit-on-error --verbose --dbname template1
"pg_upgrade_dump_13323.custom" >> "pg_upgrade_dump_13323.log" 2>&1
                                                            ok
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT d.oid, d.datname, d.encoding, d.datcollate, d.datctype,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM pg_catalog.pg_database d
 LEFT OUTER JOIN pg_catalog.pg_tablespace t  ON d.dattablespace = t.oid WHERE
d.datallowconn = true ORDER BY 2
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n          ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                                   'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR       (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c       ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;
executing: SELECT pg_catalog.set_config('search_path', '', false);
```

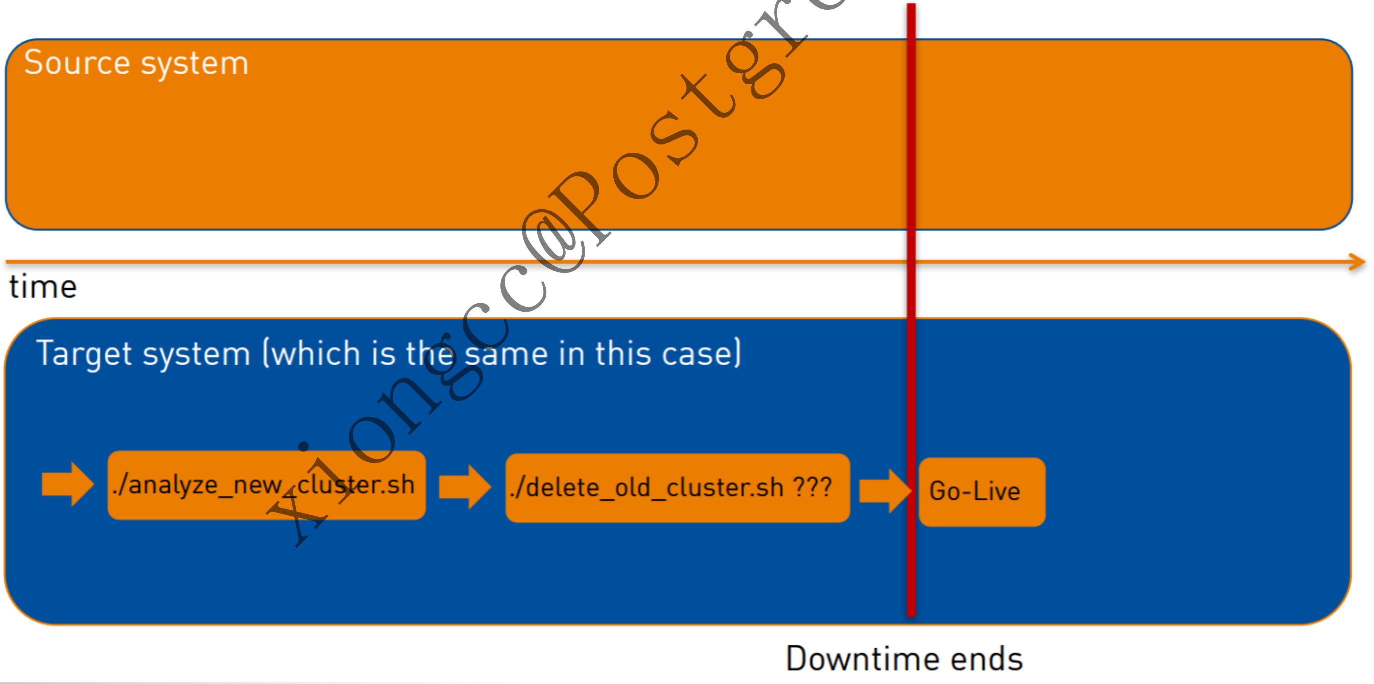
```
executing: WITH regular_heap (reloid, indtable, toastheap) AS (   SELECT c.oid, 0::oid,
0::oid   FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n          ON
c.relnamespace = n.oid   WHERE relkind IN ('r', 'm') AND     ((n.nspname !~ '^pg_temp_'
AND       n.nspname !~ '^pg_toast_temp_' AND       n.nspname NOT IN ('pg_catalog',
'information_schema',                              'binary_upgrade', 'pg_toast') AND
c.oid >= 16384::pg_catalog.oid) OR       (n.nspname = 'pg_catalog' AND       relname IN
('pg_largeobject') ))),   toast_heap (reloid, indtable, toastheap) AS (   SELECT
c.reltoastrelid, 0::oid, c.oid   FROM regular_heap JOIN pg_catalog.pg_class c       ON
regular_heap.reloid = c.oid   WHERE c.reltoastrelid != 0),   all_index (reloid,
indtable, toastheap) AS (   SELECT indexrelid, indrelid, 0::oid   FROM
pg_catalog.pg_index   WHERE indisvalid AND indisready     AND indrelid IN
(SELECT reloid FROM regular_heap          UNION ALL          SELECT reloid FROM
toast_heap)) SELECT all_rels.*, n.nspname, c.relname,   c.relfilenode, c.reltablespace,
pg_catalog.pg_tablespace_location(t.oid) AS spclocation FROM (SELECT * FROM
regular_heap       UNION ALL       SELECT * FROM toast_heap       UNION ALL
SELECT * FROM all_index) all_rels   JOIN pg_catalog.pg_class c      ON all_rels.reloid
= c.oid   JOIN pg_catalog.pg_namespace n      ON c.relnamespace = n.oid   LEFT OUTER
JOIN pg_catalog.pg_tablespace t      ON c.reltablespace = t.oid ORDER BY 1;

target databases:
Database: postgres
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:
relname: public.test: reloid: 16384 reltblspace:



Database: template1
relname: pg_catalog.pg_largeobject: reloid: 2613 reltblspace:
relname: pg_catalog.pg_largeobject_loid_pn_index: reloid: 2683 reltblspace:



"/usr/pgsql-13/bin/pg_ctl" -w -D "/home/postgres/xcc_up" -o "" -m smart stop >>
"pg_upgrade_server.log" 2>&1
Adding ".old" suffix to old global/pg_control              ok

If you want to start the old cluster, you will need to remove
the ".old" suffix from /home/postgres/9data/global/pg_control.old.
Because "link" mode was used, the old cluster cannot be safely
started once the new cluster has been started.

Linking user relation files
mappings for database "postgres":
pg_catalog.pg_largeobject: 2613 to 2613
pg_catalog.pg_largeobject_loid_pn_index: 2683 to 2683
public.test: 16384 to 16384



linking "/home/postgres/9data/base/13323/2613" to
"/home/postgres/xcc_up/base/16401/2613"
```

```
linking "/home/postgres/9data/base/13323/2683" to
"/home/postgres/xcc_up/base/16401/2683"
linking "/home/postgres/9data/base/13323/16384" to
"/home/postgres/xcc_up/base/16401/16384"
linking "/home/postgres/9data/base/13323/16384_fsm" to
"/home/postgres/xcc_up/base/16401/16384_fsm"
mappings for database "template1":
pg_catalog.pg_largeobject: 2613 to 2613
pg_catalog.pg_largeobject_loid_pn_index: 2683 to 2683


linking "/home/postgres/9data/base/1/2613" to "/home/postgres/xcc_up/base/16400/2613"
linking "/home/postgres/9data/base/1/2683" to "/home/postgres/xcc_up/base/16400/2683"
                                                            ok
Setting next OID for new cluster                            "/usr/pgsql-
13/bin/pg_resetwal" -o 16387 "/home/postgres/xcc_up" >> "pg_upgrade_utility.log" 2>&1
ok
Sync data directory to disk                                 "/usr/pgsql-13/bin/initdb"
--sync-only "/home/postgres/xcc_up" >> "pg_upgrade_utility.log" 2>&1
ok
Creating script to analyze new cluster                      ok
Creating script to delete old cluster                       ok
"/usr/pgsql-13/bin/pg_ctl" -w -l "pg_upgrade_server.log" -D "/home/postgres/xcc_up" -o
"-p 50432 -b -c synchronous_commit=off -c fsync=off -c full_page_writes=off -c
vacuum_defer_cleanup_age=0  -c listen_addresses='' -c unix_socket_permissions=0700 -c
unix_socket_directories='/home/postgres'" start >> "pg_upgrade_server.log" 2>&1
Checking for hash indexes                                   executing: SELECT
pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname FROM pg_catalog.pg_class c,
pg_catalog.pg_index i,    pg_catalog.pg_am a,     pg_catalog.pg_namespace n WHERE
i.indexrelid = c.oid AND    c.relam = a.oid AND     c.relnamespace = n.oid AND
a.amname = 'hash'
executing: SELECT pg_catalog.set_config('search_path', '', false);
executing: SELECT n.nspname, c.relname FROM pg_catalog.pg_class c,
pg_catalog.pg_index i,    pg_catalog.pg_am a,     pg_catalog.pg_namespace n WHERE
i.indexrelid = c.oid AND    c.relam = a.oid AND     c.relnamespace = n.oid AND
a.amname = 'hash'
ok
"/usr/pgsql-13/bin/pg_ctl" -w -D "/home/postgres/xcc_up" -o "" -m smart stop >>
"pg_upgrade_server.log" 2>&1

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade so,
once you start the new server, consider running:
    ./analyze_new_cluster.sh

Running this script will delete the old cluster's data files:
    ./delete_old_cluster.sh
```

如果确认升级成功，可以选择删除或者保留旧的数据库软件和集群。pg_upgrade 同样提供了一个删除旧数据库集群的脚本： ./delete_old_cluster.sh

```
rm -rf $OLDPGDATA
```


Source system
time
Target system (which is the same in this case)
./analyze_new_cluster.sh
./delete_old_cluster.sh ???


Source system
time
Target system (which is the same in this case)
./analyze_new_cluster.sh
./delete_old_cluster.sh ???
Go-Live
Downtime ends

升级之后，记得检查一下插件，必要时候可以执行update

```
alter extension xxx update;
```

关于pg_upgrade的限制，可以参照官网

> **注解**
>
> pg_upgrade创建不同的工作文件，如模式转储，在当前工作目录中。为了安全，请确保该目录不可被任何其他用户读取或者写入。
>
> pg_upgrade在新旧数据目录中启动短期的postmasters。临时 Unix 套接字文件用于与这些postmasters通信，默认情况下，在当前工作目录中进行。 在某些情况下，当前目录的路径名称可能太长，无法成为有效的套接字名称。这种情况下你可以使用`-s`选项将套接字文件放在某些具有较短路径名称的目录中。 为了安全原因，请确保该目录不可被任何其他用户读取或者写入。 （这与 Windows 无关。）
>
> 如果失败、重建和重索引会影响你的安装，pg_upgrade 将会报告这些情况。用来重建表和索引的升级后脚本将会自动被建立。 如果你正在尝试自动升级很多集簇，你应该发现具有相同数据库模式的集簇 对所有集簇升级都要求同样的升级后步骤，这是因为升级后步骤是基于数据 库模式而不是用户数据。
>
> 对于部署测试，创建一个只有模式的旧集簇副本，在其中插入假数据并且升级。
>
> pg_upgrade不支持包含使用这些`reg*` OID-引用 系统数据类型的表列的数据库的升级： `regproc`, `regprocedure`, `regoper`, `regoperator`, `regconfig`, 和 `regdictionary`. (`regtype` 能够被升级.)
>
> 如果你在升级一个PostgreSQL 9.2 之前的集簇，并且 它使用一个只有配置文件的目录，你必须向pg_upgrade 传递真正的数据目录位置，并且把配置目录位置传递给服务器，例如 `-d /real-data-directory -o '-D /configuration-directory'`。
>
> 如果正在使用的一个 9.1 之前的旧服务器用的是一个非默认Unix 域套接字目录 或者使用的默认值不同于新集簇的默认值，请把`PGHOST`设置为 指向旧服务器的套接字位置（这与 Windows 无关）。
>
> 如果你想要使用链接模式并且你不想让你的旧集簇在新集簇启动时被修改，考虑使用克隆模式。 如果(克隆模式)不可用，可以复制一份旧集簇并且在副本上以链接模式进行升级。要创建旧集簇的一 份合法拷贝，可以在服务器运行时使用`rsync`创建旧集簇的 一份脏拷贝，然后关闭旧服务器并且再次运行`rsync --checksum` 把更改更新到该拷贝以让其一致（`--checksum`是必要的，因为 `rsync`在判断文件修改时间的更改时的精度只能到秒级）。如 第 25.3.3 节中所述，你可能想要排除 一些文件，例如`postmaster.pid`。如果你的文件系统支持文 件系统快照或者 copy-on-write 文件副本，你可以使用它们来创建旧集簇和 表空间的一个备份，不过快照和副本必须被同时创建或者在数据库服务器关闭 期间被创建。

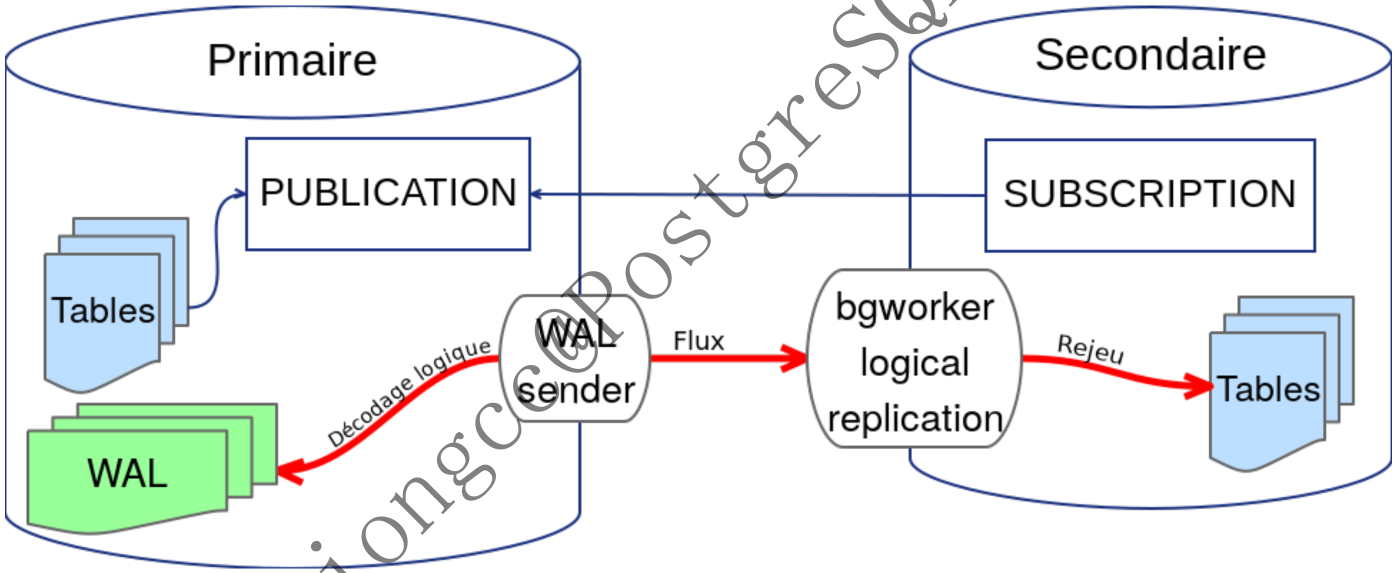
表空间的处理

表空间pg_upgrade会自动处理好，如下t1表是原来11版本，位于/home/postgres/11_tablespace的表空间下的表

```
[postgres@xiongcc ~]$ psql -p 5433
psql (14.2)
Type "help" for help.

postgres=# \d
        List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+----------
 public | t1   | table | postgres
(1 row)

postgres=# select pg_relation_filepath('t1');
            pg_relation_filepath
---------------------------------------------
 pg_tblspc/16400/PG_14_202107181/16402/16385
(1 row)

postgres=# select * from pg_tablespace ;
  oid  |  spcname   | spcowner | spcacl | spcoptions
-------+------------+----------+--------+------------
  1663 | pg_default |       10 |        |
  1664 | pg_global  |       10 |        |
 16400 | myspace    |       10 |        |
(3 rows)

postgres=# \db+
                                              List of tablespaces
    Name    |  Owner   |           Location           | Access privileges | Options |
Size    | Description
------------+----------+------------------------------+-------------------+---------+--
----------+-------------
 myspace    | postgres | /home/postgres/11_tablespace |                   |         |
4096 bytes |
 pg_default | postgres |                              |                   |         |
25 MB      |
 pg_global  | postgres |                              |                   |         |
560 kB     |
(3 rows)

[postgres@xiongcc ~]$ cd /pgdata/11_14_upgrade_data/pg_tblspc/
[postgres@xiongcc pg_tblspc]$ ll
total 0
lrwxrwxrwx 1 postgres postgres 28 Apr 20 09:29 16400 -> /home/postgres/11_tablespace
[postgres@xiongcc pg_tblspc]$ pwd
/pgdata/11_14_upgrade_data/pg_tblspc
```

logical replication

PostgreSQL 逻辑复制支持跨版本之间的数据复制，而且支持不同平台之间的复制，因此也可以用于实现版本升级。我们可以安装一个新版本的数据库作为复制的从节点，当数据已经同步时执行一次主从切换，然后关闭旧版本的主节点。主从切换的升级方法通常只需要几秒钟就能完成，利用第三方高可用组件甚至可以实现零停机时间升级。



这个就不再演示了，可以参照之前的文章 《 关于逻辑复制的方方面面》，可以使用原生的逻辑复制，也可以使用第三方扩展，如pglogical, Slony, Londiste, and Bucardo等等。也可以参照percona的例子：

1. Continuous Replication From a Legacy PostgreSQL Version to a Newer Version Using Slony
2. PostgreSQL Upgrade Using pg_dump/pg_restore
3. Replication Between PostgreSQL Versions Using Logical Replication
4. PostgreSQL Upgrade Using pg_dumpall

关于pglogical，相信也是最普遍的，使用pglogical进行升级的话，需要注意几个点，https://tech.coffeemeetsbagel.com/our-journey-to-postgresql-12-3d6ee15d305a

1. Slow synchronization can be dangerous，针对订阅端进行优化，加速订阅
    - 删除需要同步的表上的所有索引
    - 关闭 fsync
    - 将 max_wal_size 设为 50GB

    - 将 checkpoint_timeout 设为 1h

2. Every update is logged as a conflict

   当 pglologic 断定发生了冲突时，它会发出一条日志消息，比如“CONFLICT: remote UPDATE on relation PUBLIC.foo. Resolution: apply_remote”。然而，我们观察到，订阅方处理的每个更新都会被记录为冲突。仅经过几个小时的复制，订阅方数据库就已经写入了 1GB 的冲突日志。设置pglogical.conflict_log_level = DEBUG 将其关闭，

集群升级

使用link模式升级之后，原来的standby日志种会打印如下错误日志

system identifier的作用：

1. 当使用流复制的物理备库时，需要判断上下游节点的system id是否一致，如果不一致，物理复制中断（当然，物理备库是完全一致的，因为文件级一致）。
2. 在recovery时，如果发现xlog的systemid与当前数据库的systemid不一致，同样也会不使用这个xlog文件。这个目的当然也很纯洁，因为只会使用自己产生的xlog，当然不能用别人（的库）产生的xlog，避免用错xlog。

```
2021-07-06 23:06:32.152 CST,,,19316,,60e47178.4b74,1,,2021-07-06 23:06:32 CST,,0,FATAL,XX000,"database system identifier differs between the primary and standby","The primary's identifier is 6981828042669287918, the standby's identifier is 6981826014922692999.",,,,,,,,,""
2021-07-06 23:06:37.156 CST,,,19320,,60e4717d.4b78,1,,2021-07-06 23:06:37 CST,,0,FATAL,XX000,"database system identifier differs between the primary and standby","The primary's identifier is 6981828042669287918, the standby's identifier is 6981826014922692999.",,,,,,,,,""
```

直接对备库进行依葫芦画瓢使用pg_upgrade升级是不行的

```
[postgres@xiongcc ~]$ pg_upgrade --old-datadir /home/postgres/12data_bak/ --new-datadir /home/postgres/upgrade_test_bak/ --old-bindir=/usr/pgsql-12/bin/ --new-bindir=/usr/pgsql-13/bin/ --check
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok

The source cluster was shut down while in recovery mode.  To upgrade, use "rsync" as documented or shut it down as a primary.
Failure, exiting
```

假如promote为了新主再使用pg_upgrade升级，就会回到之前的错误了

```
2021-07-06 23:32:14.103 CST [20430] FATAL:  database system identifier differs between the primary and standby
2021-07-06 23:32:14.103 CST [20430] DETAIL:  The primary's identifier is 6972099375733193972, the standby's identifier is 6981835965031898837.
2021-07-06 23:32:14.106 CST [20432] FATAL:  database system identifier differs between the primary and standby
2021-07-06 23:32:14.106 CST [20432] DETAIL:  The primary's identifier is 6972099375733193972, the standby's identifier is 6981835965031898837.
```

所以，升级standby的办法是：

1. 简单粗暴的重新搭建备机，不过对于大库来说比较费力
2. 使用官方的rsync方式

```
rsync --archive --delete --hard-links --size-only --no-inc-recursive
/opt/PostgreSQL/9.5 \
      /opt/PostgreSQL/9.6 standby.example.com:/opt/PostgreSQL
```

示例如下

1. 12data是升级前的12版本的master
2. upgrade_test是升级后的13版本
3. /home/postgres/upgrade_test_bak/是升级后的13版本standby

```
[postgres@xiongcc ~]$ rsync --archive --delete --hard-links --size-only --no-inc-recursive /home/postgres/12data /home/postgres/upgrade_test
/home/postgres/upgrade_test_bak/
```

对比

1. dump/dumpall + restore/psql：由于dump+restore总体相对耗时较长，因此不适用于大数据量的数据库，或是写入比较频繁的场景使用。尤其是对于核心的业务程序（特别是有99.999保证的程序）来说是不可接受的，好处就是十分安全，当然也可以不停机，使用逻辑复制搭配decoder_raw等Plugins解析出备份之后的操作SQL，不过不能保证一致性。
2. pg_upgrade：此方案优势是速度非常快，搭配并行和link模式，pg_upgrade –link –jobs xxx，但是必须停机升级，并且link模式，会无法使用以前的版本，因此不适用于7x24的场景，若使用默认模式，就地升级会导致磁盘使用率增倍，好处就是源端还可以用，同时假如还有standby或者流复制的场景下，可能还会丢失备库，可以使用官方推荐的rsync模式。同时值得注意的是，升级过程目前不会复制数据分布的任何统计数据，比如像直方图、最常见的值及其频率之类的东西，所以这也是“vacuumdb –analyze-only –analyze-in-stage”的重要性，不然复杂查询的性能会受到相当大的影响，还有就是make 、configure编译的参数得保持一致
3. 逻辑复制：此方案是最平滑的方案，比较适用于7x24小时以及大数据量场景，停机时间非常短，只有几秒钟，同时源端和目标端升级之后假如有需求，还是可以使用的。但是配置繁琐，尤其是v10以前的版本，还没有原生逻辑复制，需要第三方扩展，第三方扩展除了问题还是黑盒，增加了运维成本，同时还要有集群环境，但是原生逻辑复制又不支持序列、DDL（搭配插件pg_ddl_deploy，不过又回到了黑盒来了）、视图等等。

Upgrade method	Pro	Contra
Dump / restore	• Simple • Safe • Somewhat flexible	• Slowest method • Per database approach has some pitfalls lurking
Binary in-place	• Fast / very fast (depending on chosen mode) • Old instance not affected in default mode	• More complex than Dump / Restore • Somewhat risky in “link” mode • Possibly loses standby servers • Double the disk space required in default mode
Logical Replication	• Shortest possible downtime • Safe, with possibility of thorough “live tests” • Very flexible	• Most complex method • Possibly some schema changes needed • Not everything is transferred (sequence state, large objects) • Possibly “slowish” • Always per database

关于pg_upgrade的升级速度可以查看：How fast is pg_upgrade anyway? https://www.endpoint.com/blog/2015/07/01/how-fast-is-pgupgrade-anyway

参考

https://www.cybertec-postgresql.com/en/a-primer-on-postgresql-upgrade-methods/

https://blog.crunchydata.com/blog/performing-a-major-postgresql-upgrade-with-pg_dumpall

https://tech.coffeemeetsbagel.com/our-journey-to-postgresql-12-3d6ee15d305a

https://www.percona.com/blog/2019/04/12/fast-upgrade-of-legacy-postgresql-with-minimum-downtime-using-pg_upgrade/

https://www.endpoint.com/blog/2015/07/01/how-fast-is-pgupgrade-anyway

https://www.pgday.ch/common/slides/2017_T-DBI-20170630-001-PGDAY-Upgrade-BestPractices.pdf

https://www.cybertec-postgresql.com/en/upgrading-and-updating-postgresql/

https://www.postgresql.org/docs/current/pgupgrade.html

https://why-upgrade.depesz.com/show?from=13.1&to=13.3&keywords==-pooo0