check_postgres.pl> :: check_postgres

Source

License

View All Documentation

- PDF
- EPUB
- Home

Navigation :

navigation

check_postgres

- dbstats
- disabled_triggers
- disk_space
- fsm_pages
- fsm_relations
- hitratio
- hot_standby_delay
- relation_size
- index_size
- table_size
- indexes_size
- total_relation_size
- last_analyze
- last_vacuum
- last_autoanalyze
- last_autovacuum
- listener
- locks
- logfile
- new_version_bc
- new_version_box
- new_version_cp
- new_version_pg
- new_version_tnm
- pgb_pool_cl_active
- pgb_pool_cl_waiting
- pgb_pool_sv_active
- pgb_pool_sv_idle
- pgb_pool_sv_used
- pgb_pool_sv_tested
- pgb_pool_sv_login
- pgb_pool_maxwait
- pgbouncer_backends
- pgbouncer_checksum
- pgbouncer_maxwait
- pgagent_jobs
- prepared_txns
- query_runtime
- query_time
- replicate_row
- replication_slots
- same_schema
- sequence
- settings_checksum
- slony_status
- timesync

check_postgres.pl - a Postgres monitoring script for Nagios, MRTG, Cacti, and others

This documents describes check_postgres.pl version 2.26.0

SYNOPSIS

```
  ## Create all symlinks
  check_postgres.pl -symlinks

## Check connection to Postgres database 'pluto':
  check_postgres.pl -action=connection -db=pluto

## Same things, but using the symlink
  check_postgres_connection -db=pluto

## Warn if > 100 locks, critical if > 200, or > 20 exclusive
  check_postgres_locks -warning=100 -critical="total=200:exclusive=20"

## Show the current number of idle connections on port 6543:
  check_postgres_txn_idle -port=6543 -output=simple

## There are many other actions and options, please keep reading.

The latest news and documentation can always be found at:
```

`https://bucardo.org/check_postgres/`

DESCRIPTION

check_postgres.pl is a Perl script that runs many different tests against one or more Postgres databases. It uses the psql program to gather the information, and outputs the results in one of three formats: Nagios, MRTG, or simple.

Output Modes

The output can be changed by use of the `-output` option. The default output is nagios, although this can be changed at the top of the script if you wish. The current option choices are **nagios** , **mrtg** , and **simple** . To avoid having to enter the output argument each time, the type of output is automatically set if no –output argument is given, and if the current directory has one of the output options in its name. For example, creating a directory named mrtg and populating it with symlinks via the *–symlinks* argument would ensure that any actions run from that directory will always default to an output of "mrtg" As a shortcut for –output=simple, you can enter –simple, which also overrides the directory naming trick.

Nagios output The default output format is for Nagios, which is a single line of information, along with four specific exit codes:

0 (OK)
1 (WARNING)
2 (CRITICAL)
3 (UNKNOWN)

The output line is one of the words above, a colon, and then a short description of what was measured. Additional statistics information, as well as the total time the command took, can be output as well: see the documentation on the arguments *–showperf* , *–perflimit* , and *–showtime* .

MRTG output The MRTG output is four lines, with the first line always giving a single number of importance. When possible, this number represents an actual value such as a number of bytes, but it may also be a 1 or a 0 for actions that only return "true" or "false", such as check_postgres_version. The second line is an additional stat and is only used for some actions. The third line indicates an "uptime" and is not used. The fourth line is a description and usually indicates the name of the database the stat from the first line was pulled from, but may be different depending on the action.

Some actions accept an optional *–mrtg* argument to further control the output.

See the documentation on each action for details on the exact MRTG output for each one.

Simple output The simple output is simply a truncated version of the MRTG one, and simply returns the first number and nothing else. This is very useful when you just want to check the state of something, regardless of any threshold. You can transform the numeric output by appending KB, MB, GB, TB, or EB to the output argument, for example:

```
-output=simple,MB
```

Cacti output The Cacti output consists of one or more items on the same line, with a simple name, a colon, and then a number. At the moment, the only action with explicit Cacti output is 'dbstats', and using the –output option is not needed in this case, as Cacti is the only output for this action. For many other actions, using –simple is enough to make Cacti happy.

DATABASE CONNECTION OPTIONS

All actions accept a common set of database options.

-H NAME or –host=NAME Connect to the host indicated by NAME. Can be a comma-separated list of names. Multiple host arguments are allowed. If no host is given, defaults to the `PGHOST` environment variable or no host at all (which indicates using a local Unix socket). You may also use "–dbhost".

-p PORT or –port=PORT Connects using the specified PORT number. Can be a comma-separated list of port numbers, and multiple port arguments are allowed. If no port number is given, defaults to the `PGPORT` environment variable. If that is not set, it defaults to 5432. You may also use "–dbport"

-db NAME or –dbname=NAME Specifies which database to connect to. Can be a comma-separated list of names, and multiple dbname arguments are allowed. If no dbname option is provided, defaults to the `PGDATABASE` environment variable. If that is not set, it defaults to 'postgres' if psql is version 8 or greater, and 'template1' otherwise.

-u USERNAME or –dbuser=USERNAME The name of the database user to connect as. Can be a comma-separated list of usernames, and multiple dbuser arguments are allowed. If this is not provided, it defaults to the `PGUSER` environment variable, otherwise it defaults to 'postgres'.

–dbpass=PASSWORD Provides the password to connect to the database with. Use of this option is highly discouraged. Instead, one should use a .pgpass or pg_service.conf file.

–dbservice=NAME The name of a service inside of the pg_service.conf file. Before version 9.0 of Postgres, this is a global file, usually found in */etc/pg_service.conf* . If you are using version 9.0 or higher of Postgres,

you can use the file ".pg_service.conf" in the home directory of the user running the script, e.g. nagios.

This file contains a simple list of connection options. You can also pass additional information when using this option such as –dbservice="maindatabase sslmode=require"

The documentation for this file can be found at https://www.postgresql.org/docs/current/static/libpq-pgservice.html

–role=ROLE Provides the role to switch to after connecting to the database but before running the given check. This provides the ability to have superuser privileges assigned to a role without LOGIN access for the purposes of audit and other security considerations. Requires a local `psql` version 9.6 or higher.

The database connection options can be grouped: *–host=a,b –host=c –port=1234 –port=3344* would connect to a-1234, b-1234, and c-3344. Note that once set, an option carries over until it is changed again.

Examples:

```
  -host=a,b -port=5433 -db=c
  Connects twice to port 5433, using database c, to hosts a and b: a-5433-c b-5433-c

-host=a,b -port=5433 -db=c,d
  Connects four times: a-5433-c a-5433-d b-5433-c b-5433-d

-host=a,b -host=foo -port=1234 -port=5433 -db=e,f
  Connects six times: a-1234-e a-1234-f b-1234-e b-1234-f foo-5433-e foo-5433-f

-host=a,b -host=x -port=5432,5433 -dbuser=alice -dbuser=bob -db=baz
  Connects three times: a-5432-alice-baz b-5433-alice-baz x-5433-bob-baz

-dbservice="foo" -port=5433
  Connects using the named service 'foo' in the pg_service.conf file, but overrides the port
```

OTHER OPTIONS

Other options include:

–action=NAME States what action we are running. Required unless using a symlinked file, in which case the name of the file is used to figure out the action.

–warning=VAL or -w VAL Sets the threshold at which a warning alert is fired. The valid options for this option depends on the action used.

–critical=VAL or -c VAL Sets the threshold at which a critical alert is fired. The valid options for this option depends on the action used.

-t VAL or –timeout=VAL Sets the timeout in seconds after which the script will abort whatever it is doing and return an UNKNOWN status. The timeout is per Postgres cluster, not for the entire script. The default value is 10; the units are always in seconds.

–**assume-standby-mode** If specified, first the check if server in standby mode will be performed (–datadir is required), if so, all checks that require SQL queries will be ignored and "Server in standby mode" with OK status will be returned instead.

Example:

```
postgres@db$./check_postgres.pl -action=version -warning=8.1 -datadir /var/lib/post
POSTGRES_VERSION OK:  Server in standby mode | time=0.00
```

–**assume-prod** If specified, check if server in production mode is performed (–datadir is required). The option is only relevant for (`symlink: check_postgres_checkpoint`).

Example:

```
postgres@db$./check_postgres.pl -action=checkpoint -datadir /var/lib/postgresql/8.3
POSTGRES_CHECKPOINT OK: Last checkpoint was 72 seconds ago | age=72;;300 mode=MASTE
```

–**assume-async** If specified, indicates that any replication between servers is asynchronous. The option is only relevant for (`symlink: check_postgres_same_schema`).

Example: postgres@db$./check_postgres.pl –action=same_schema –assume-async –dbhost=star,line

-h or –help Displays a help screen with a summary of all actions and options.

–**man** Displays the entire manual.

-V or –version Shows the current version.

-v or –verbose Set the verbosity level. Can call more than once to boost the level. Setting it to three or higher (in other words, issuing `-v -v -v`) turns on debugging information for this program which is sent to stderr.

–**showperf=VAL** Determines if we output additional performance data in standard Nagios format (at end of string, after a pipe symbol, using name=value). VAL should be 0 or 1. The default is 1. Only takes effect if using Nagios output mode.

–**perflimit=i** Sets a limit as to how many items of interest are reported back when using the *showperf* option. This only has an effect for actions that return a large number of items, such as **table_size** . The default is 0, or no limit. Be careful when using this with the *–include* or *–exclude* options, as those restrictions are done *after* the query has been run, and thus your limit may not include the items you want. Only takes effect if using Nagios output mode.

–**showtime=VAL** Determines if the time taken to run each query is shown in the output. VAL should be 0 or 1. The default is 1. No effect unless *showperf* is on. Only takes effect if using Nagios output mode.

–**test** Enables test mode. See the "TEST MODE" section below.

–**PGBINDIR=PATH** Tells the script where to find the psql binaries. Useful if you have more than one version of the PostgreSQL executables on your system, or if there are not in your path. Note that this option is in all uppercase. By default, this option is *not allowed* . To enable it, you must change the `$NO_PSQL_OPTION` near the top of the script to 0. Avoid using this option if you can, and instead use environment variable c or hard-coded `$PGBINDIR` variable, also near the top of the script, to set the path to the PostgreSQL to use.

–**PSQL=PATH** *(deprecated, this option may be removed in a future release!)* Tells the script where to find the psql program. Useful if you have more than one version of the psql executable on your system, or if there is no psql program in your path. Note that this option is in all uppercase. By default, this option is *not allowed* . To enable it, you must change the `$NO_PSQL_OPTION` near the top of the script to 0. Avoid using this option if you can, and instead hard-code your psql location into the `$PSQL` variable, also near the top of the script.

–**symlinks** Creates symlinks to the main program for each action.

–**output=VAL** Determines the format of the output, for use in various programs. The default is 'nagios'. Available options are 'nagios', 'mrtg', 'simple' and 'cacti'.

–**mrtg=VAL** Used only for the MRTG or simple output, for a few specific actions.

–**debugoutput=VAL** Outputs the exact string returned by psql, for use in debugging. The value is one or more letters, which determine if the output is displayed or not, where 'a' = all, 'c' = critical, 'w' = warning, 'o' = ok, and 'u' = unknown. Letters can be combined.

–**get_method=VAL** Allows specification of the method used to fetch information for the `new_version_cp` , `new_version_pg` , `new_version_bc` , `new_version_box` , and `new_version_tnm` checks. The following programs are tried, in order, to grab the information from the web: GET, wget, fetch, curl, lynx, links. To force the use of just one (and thus remove the overhead of trying all the others until one of those works), enter one of the names as the argument to get_method. For example, a BSD box might enter the following line in their `.check_postgresrc` file:

```
get_method=fetch
```

–**language=VAL** Set the language to use for all output messages. Normally, this is detected by examining the environment variables LC_ALL, LC_MESSAGES, and LANG, but setting this option will override any such detection.

ACTIONS

The action to be run is selected using the –action flag, or by using a symlink to the main file that contains the name of the action inside of it. For example, to run the action "timesync", you may either issue:

```
check_postgres.pl -action=timesync
```

or use a program named:

```
check_postgres_timesync
```

All the symlinks are created for you in the current directory if use the option –symlinks:

```
perl check_postgres.pl -symlinks
```

If the file name already exists, it will not be overwritten. If the file exists and is a symlink, you can force it to overwrite by using "–action=build_symlinks_force".

Most actions take a *–warning* and a *–critical* option, indicating at what point we change from OK to WARNING, and what point we go to CRITICAL. Note that because criticals are always checked first, setting the warning equal to the critical is an effective way to turn warnings off and always give a critical.

The current supported actions are:

archive_ready

(`symlink: check_postgres_archive_ready`) Checks how many WAL files with extension *.ready* exist in the *pg_xlog/archive_status* directory (PostgreSQL 10 and later: *pg_wal/archive_status*), which is found off of your **data_directory** . If the *–lsfunc* option is not used then this action must be run as a superuser, in order to access the contents of the *pg_xlog/archive_status* directory. The minimum version to use this action is Postgres 8.1. The *–warning* and *–critical* options are simply the number of *.ready* files in the *pg_xlog/archive_status* directory. Usually, these values should be low, turning on the archive mechanism, we usually want it to archive WAL files as fast as possible.

If the archive command fail, number of WAL in your *pg_xlog* directory will grow until exhausting all the disk space and force PostgreSQL to stop immediately.

To avoid running as a database superuser, a wrapper function around `pg_ls_dir()` should be defined as a superuser with SECURITY DEFINER, and the *–lsfunc* option used. This example function, if defined by

a superuser, will allow the script to connect as a normal user *nagios* with *–lsfunc=ls_archive_status_dir*

```
BEGIN;
CREATE FUNCTION ls_archive_status_dir()
    RETURNS SETOF TEXT
    AS $$ SELECT pg_ls_dir('pg_xlog/archive_status') $$
    LANGUAGE SQL
    SECURITY DEFINER;
REVOKE ALL ON FUNCTION ls_archive_status_dir() FROM PUBLIC;
GRANT EXECUTE ON FUNCTION ls_archive_status_dir() to nagios;
COMMIT;
```

Example 1: Check that the number of ready WAL files is 10 or less on host "pluto", using a wrapper function `ls_archive_status_dir` to avoid the need for superuser permissions

```
check_postgres_archive_ready -host=pluto -critical=10 -lsfunc=ls_archive_status_dir
```

For MRTG output, reports the number of ready WAL files on line 1.

autovac_freeze

(`symlink: check_postgres_autovac_freeze`) Checks how close each database is to the Postgres **autovacuum_freeze_max_age** setting. This action will only work for databases version 8.2 or higher. The *–warning* and *–critical* options should be expressed as percentages. The 'age' of the transactions in each database is compared to the autovacuum_freeze_max_age setting (200 million by default) to generate a rounded percentage. The default values are **90%** for the warning and **95%** for the critical. Databases can be filtered by use of the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details.

Example 1: Give a warning when any databases on port 5432 are above 97%

```
check_postgres_autovac_freeze -port=5432 -warning="97%"
```

For MRTG output, the highest overall percentage is reported on the first line, and the highest age is reported on the second line. All databases which have the percentage from the first line are reported on the fourth line, separated by a pipe symbol.

backends

(`symlink: check_postgres_backends`) Checks the current number of connections for one or more databases, and optionally compares it to the maximum allowed, which is determined by the Postgres configuration variable **max_connections** . The *–warning* and *–critical* options can take one of three forms. First, a simple number can be given, which represents the number of connections at which the alert will be given. This choice does not use the

max_connections setting. Second, the percentage of available connections can be given. Third, a negative number can be given which represents the number of connections left until **max_connections** is reached. The default values for *–warning* and *–critical* are '90%' and '95%'. You can also filter the databases by use of the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details.

To view only non-idle processes, you can use the *–noidle* argument. Note that the user you are running as (either connecting directly or switching via *–role*) must be a superuser for this to work properly.

Example 1: Give a warning when the number of connections on host quirm reaches 120, and a critical if it reaches 150.

```
check_postgres_backends -host=quirm -warning=120 -critical=150
```

Example 2: Give a critical when we reach 75% of our max_connections setting on hosts lancre or lancre2.

```
check_postgres_backends -warning='75%' -critical='75%' -host=lancre,lancre2
```

Example 3: Give a warning when there are only 10 more connection slots left on host plasmid, and a critical when we have only 5 left.

```
check_postgres_backends -warning=-10 -critical=-5 -host=plasmid
```

Example 4: Check all databases except those with "test" in their name, but allow ones that are named "pg_greatest". Connect as port 5432 on the first two hosts, and as port 5433 on the third one. We want to always throw a critical when we reach 30 or more connections.

```
check_postgres_backends -dbhost=hong,kong -dbhost=fooey -dbport=5432 -dbport=5433 -warning=
```

For MRTG output, the number of connections is reported on the first line, and the fourth line gives the name of the database, plus the current maximum_connections. If more than one database has been queried, the one with the highest number of connections is output.

bloat

(`symlink: check_postgres_bloat`) Checks the amount of bloat in tables and indexes. (Bloat is generally the amount of dead unused space taken up in a table or index. This space is usually reclaimed by use of the VACUUM command.) This action requires that stats collection be enabled on the target databases, and requires that ANALYZE is run frequently. The *–include* and *–exclude* options can be used to filter out which tables to look at. See the "BASIC FILTERING" section for more details.

The *–warning* and *–critical* options can be specified as sizes, percents, or both. Valid size units are bytes, kilobytes, megabytes, gigabytes, terabytes, exabytes, petabytes, and zettabytes. You can abbreviate all of those with the first letter.

Items without units are assumed to be 'bytes'. The default values are '1 GB' and '5 GB'. The value represents the number of "wasted bytes", or the difference between what is actually used by the table and index, and what we compute that it should be.

Note that this action has two hard-coded values to avoid false alarms on smaller relations. Tables must have at least 10 pages, and indexes at least 15, before they can be considered by this test. If you really want to adjust these values, you can look for the variables *$MINPAGES* and *$MINIPAGES* at the top of the `check_bloat` subroutine. These values are ignored if either *–exclude* or *–include* is used.

Only the top 10 most bloated relations are shown. You can change this number by using the *–perflimit* option to set your own limit.

The schema named 'information_schema' is excluded from this test, as the only tables it contains are small and do not change.

Please note that the values computed by this action are not precise, and should be used as a guideline only. Great effort was made to estimate the correct size of a table, but in the end it is only an estimate. The correct index size is even more of a guess than the correct table size, but both should give a rough idea of how bloated things are.

Example 1: Warn if any table on port 5432 is over 100 MB bloated, and critical if over 200 MB

```
check_postgres_bloat -port=5432 -warning='100 M' -critical='200 M'
```

Example 2: Give a critical if table 'orders' on host 'sami' has more than 10 megs of bloat

```
check_postgres_bloat -host=sami -include=orders -critical='10 MB'
```

Example 3: Give a critical if table 'q4' on database 'sales' is over 50% bloated

```
check_postgres_bloat -db=sales -include=q4 -critical='50%'
```

Example 4: Give a critical any table is over 20% bloated *and* has over 150 MB of bloat:

```
check_postgres_bloat -port=5432 -critical='20% and 150 M'
```

Example 5: Give a critical any table is over 40% bloated *or* has over 500 MB of bloat:

```
check_postgres_bloat -port=5432 -warning='500 M or 40%'
```

For MRTG output, the first line gives the highest number of wasted bytes for the tables, and the second line gives the highest number of wasted bytes for the indexes. The fourth line gives the database name, table name, and index name information. If you want to output the bloat ratio instead (how many

times larger the relation is compared to how large it should be), just pass in `-mrtg=ratio` .

checkpoint

(`symlink: check_postgres_checkpoint`) Determines how long since the last checkpoint has been run. This must run on the same server as the database that is being checked (e.g. the -h flag will not work). This check is meant to run on a "warm standby" server that is actively processing shipped WAL files, and is meant to check that your warm standby is truly 'warm'. The data directory must be set, either by the environment variable `PGDATA` , or passing the `-datadir` argument. It returns the number of seconds since the last checkpoint was run, as determined by parsing the call to `pg_controldata` . Because of this, the pg_controldata executable must be available in the current path. Alternatively, you can specify `PGBINDIR` as the directory that it lives in. It is also possible to use the special options *–assume-prod* or *–assume-standby-mode* , if the mode found is not the one expected, a CRITICAL is emitted.

At least one warning or critical argument must be set.

This action requires the Date::Parse module.

For MRTG or simple output, returns the number of seconds.

cluster_id

(`symlink: check_postgres_cluster-id`) Checks that the Database System Identifier provided by pg_controldata is the same as last time you checked. This must run on the same server as the database that is being checked (e.g. the -h flag will not work). Either the *–warning* or the *–critical* option should be given, but not both. The value of each one is the cluster identifier, an integer value. You can run with the special `-critical=0` option to find out an existing cluster identifier.

Example 1: Find the initial identifier

```
check_postgres_cluster_id -critical=0 -datadir=/var//lib/postgresql/9.0/main
```

Example 2: Make sure the cluster is the same and warn if not, using the result from above.

```
check_postgres_cluster_id  -critical=5633695740047915135
```

For MRTG output, returns a 1 or 0 indicating success of failure of the identifier to match. A identifier must be provided as the `-mrtg` argument. The fourth line always gives the current identifier.

commitratio

(`symlink: check_postgres_commitratio`) Checks the commit ratio of all databases and complains when they are too low. There is no need to run this

command more than once per database cluster. Databases can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. They can also be filtered by the owner of the database with the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The warning and critical options should be specified as percentages. There are not defaults for this action: the warning and critical must be specified. The warning value cannot be greater than the critical value. The output returns all databases sorted by commitratio, smallest first.

Example: Warn if any database on host flagg is less than 90% in commitratio, and critical if less then 80%.

```
check_postgres_database_commitratio -host=flagg -warning='90%' -critical='80%'
```

For MRTG output, returns the percentage of the database with the smallest commitratio on the first line, and the name of the database on the fourth line.

connection

(`symlink: check_postgres_connection`) Simply connects, issues a 'SELECT version()', and leaves. Takes no *–warning* or *–critical* options.

For MRTG output, simply outputs a 1 (good connection) or a 0 (bad connection) on the first line.

custom_query

(`symlink: check_postgres_custom_query`) Runs a custom query of your choosing, and parses the results. The query itself is passed in through the `query` argument, and should be kept as simple as possible. If at all possible, wrap it in a view or a function to keep things easier to manage. The query should return one or two columns. It is required that one of the columns be named "result" and is the item that will be checked against your warning and critical values. The second column is for the performance data and any name can be used: this will be the 'value' inside the performance data section.

At least one warning or critical argument must be specified. What these are set to depends on the type of query you are running. There are four types of custom_queries that can be run, specified by the `valtype` argument. If none is specified, this action defaults to 'integer'. The four types are:

integer : Does a simple integer comparison. The first column should be a simple integer, and the warning and critical values should be the same.

string : The warning and critical are strings, and are triggered only if the value in the first column matches it exactly. This is case-sensitive.

time : The warning and the critical are times, and can have units of seconds, minutes, hours, or days. Each may be written singular or abbreviated to just

the first letter. If no units are given, seconds are assumed. The first column should be an integer representing the number of seconds to check.

size : The warning and the critical are sizes, and can have units of bytes, kilobytes, megabytes, gigabytes, terabytes, or exabytes. Each may be abbreviated to the first letter. If no units are given, bytes are assumed. The first column should be an integer representing the number of bytes to check.

Normally, an alert is triggered if the values returned are **greater than** or equal to the critical or warning value. However, an option of *–reverse* will trigger the alert if the returned value is **lower than** or equal to the critical or warning value.

Example 1: Warn if any relation over 100 pages is named "rad", put the number of pages inside the performance data section.

```
check_postgres_custom_query -valtype=string -w "rad" -query=
  "SELECT relname AS result, relpages AS pages FROM pg_class WHERE relpages > 100"
```

Example 2: Give a critical if the "foobar" function returns a number over 5MB:

```
check_postgres_custom_query -critical='5MB'-valtype=size -query="SELECT foobar() AS result
```

Example 2: Warn if the function "snazzo" returns less than 42:

```
check_postgres_custom_query -critical=42 -query="SELECT snazzo() AS result" -reverse
```

If you come up with a useful custom_query, consider sending in a patch to this program to make it into a standard action that other people can use.

This action does not support MRTG or simple output yet.

database_size

(`symlink: check_postgres_database_size`) Checks the size of all databases and complains when they are too big. There is no need to run this command more than once per database cluster. Databases can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. They can also be filtered by the owner of the database with the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The warning and critical options can be specified as bytes, kilobytes, megabytes, gigabytes, terabytes, or exabytes. Each may be abbreviated to the first letter as well. If no unit is given, the units are assumed to be bytes. There are not defaults for this action: the warning and critical must be specified. The warning value cannot be greater than the critical value. The output returns all databases sorted by size largest first, showing both raw bytes and a "pretty" version of the size.

Example 1: Warn if any database on host flagg is over 1 TB in size, and critical if over 1.1 TB.

```
check_postgres_database_size -host=flagg -warning='1 TB' -critical='1.1 t'
```

Example 2: Give a critical if the database template1 on port 5432 is over 10 MB.

```
check_postgres_database_size -port=5432 -include=template1 -warning='10MB' -critical='10M
```

Example 3: Give a warning if any database on host 'tardis' owned by the user 'tom' is over 5 GB

```
check_postgres_database_size -host=tardis -includeuser=tom -warning='5 GB' -critical='10 G
```

For MRTG output, returns the size in bytes of the largest database on the first line, and the name of the database on the fourth line.

dbstats

(`symlink: check_postgres_dbstats`) Reports information from the pg_stat_database view, and outputs it in a Cacti-friendly manner. No other output is supported, as the output is informational and does not lend itself to alerts, such as used with Nagios. If no options are given, all databases are returned, one per line. You can include a specific database by use of the `-include` option, or you can use the `-dbname` option.

Eleven items are returned on each line, in the format name:value, separated by a single space. The items are:

backends The number of currently running backends for this database.

commits The total number of commits for this database since it was created or reset.

rollbacks The total number of rollbacks for this database since it was created or reset.

read The total number of disk blocks read.

hit The total number of buffer hits.

ret The total number of rows returned.

fetch The total number of rows fetched.

ins The total number of rows inserted.

upd The total number of rows updated.

del The total number of rows deleted.

dbname The name of the database.

Note that ret, fetch, ins, upd, and del items will always be 0 if Postgres is version 8.2 or lower, as those stats were not available in those versions.

If the dbname argument is given, seven additional items are returned:

idxscan Total number of user index scans.

idxtupread Total number of user index entries returned.

idxtupfetch Total number of rows fetched by simple user index scans.

idxblksread Total number of disk blocks read for all user indexes.

idxblkshit Total number of buffer hits for all user indexes.

seqscan Total number of sequential scans against all user tables.

seqtupread Total number of tuples returned from all user tables.

Example 1: Grab the stats for a database named "products" on host "willow":

```
check_postgres_dbstats -dbhost willow -dbname products
```

The output returned will be like this (all on one line, not wrapped):

```
backends:82 commits:58374408 rollbacks:1651 read:268435543 hit:2920381758 idxscan:310931
idxtupfetch:1840241349 idxblksread:62860110 idxblkshit:1107812216 seqscan:5085305 seqtup
ret:0 fetch:0 ins:0 upd:0 del:0 dbname:willow
```

disabled_triggers

(`symlink: check_postgres_disabled_triggers`) Checks on the number of disabled triggers inside the database. The *–warning* and *–critical* options are the number of such triggers found, and both default to "1", as in normal usage having disabled triggers is a dangerous event. If the database being checked is 8.3 or higher, the check is for the number of triggers that are in a 'disabled' status (as opposed to being 'always' or 'replica'). The output will show the name of the table and the name of the trigger for each disabled trigger.

Example 1: Make sure that there are no disabled triggers

```
check_postgres_disabled_triggers
```

For MRTG output, returns the number of disabled triggers on the first line.

disk_space

(`symlink: check_postgres_disk_space`) Checks on the available physical disk space used by Postgres. This action requires that you have the executable "/bin/df" available to report on disk sizes, and it also needs to be run as a superuser (either connecting directly or switching via *–role*), so it can examine the **data_directory** setting inside of Postgres. The *–warning* and *–critical* options are given in either sizes or percentages or both. If using sizes, the standard unit types are allowed: bytes, kilobytes, gigabytes, megabytes, gigabytes, terabytes, or exabytes. Each may be abbreviated to the first letter only; no units at all indicates 'bytes'. The default values are '90%' and '95%'.

This command checks the following things to determine all of the different physical disks being used by Postgres.

data_directory - The disk that the main data directory is on.

log directory - The disk that the log files are on.

WAL file directory - The disk that the write-ahead logs are on (e.g. symlinked pg_xlog or pg_wal)

tablespaces - Each tablespace that is on a separate disk.

The output shows the total size used and available on each disk, as well as the percentage, ordered by highest to lowest percentage used. Each item above maps to a file system: these can be included or excluded. See the "BASIC FILTERING" section for more details.

Example 1: Make sure that no file system is over 90% for the database on port 5432.

```
check_postgres_disk_space -port=5432 -warning='90%' -critical='90%'
```

Example 2: Check that all file systems starting with */dev/sda* are smaller than 10 GB and 11 GB (warning and critical)

```
check_postgres_disk_space -port=5432 -warning='10 GB' -critical='11 GB' -include="~^/dev/s
```

Example 4: Make sure that no file system is both over 50% *and* has over 15 GB

```
check_postgres_disk_space -critical='50% and 15 GB'
```

Example 5: Issue a warning if any file system is either over 70% full *or* has more than 1T

```
check_postgres_disk_space -warning='1T or 75'
```

For MRTG output, returns the size in bytes of the file system on the first line, and the name of the file system on the fourth line.

fsm_pages

(`symlink: check_postgres_fsm_pages`) Checks how close a cluster is to the Postgres **max_fsm_pages** setting. This action will only work for databases of 8.2 or higher, and it requires the contrib module **pg_freespacemap** be installed. The *–warning* and *–critical* options should be expressed as percentages. The number of used pages in the free-space-map is determined by looking in the pg_freespacemap_relations view, and running a formula based on the formula used for outputting free-space-map pageslots in the vacuum verbose command. The default values are **85%** for the warning and **95%** for the critical.

Example 1: Give a warning when our cluster has used up 76% of the free-space pageslots, with pg_freespacemap installed in database robert

```
check_postgres_fsm_pages -dbname=robert -warning="76%"
```

While you need to pass in the name of the database where pg_freespacemap is installed, you only need to run this check once per cluster. Also, checking this

information does require obtaining special locks on the free-space-map, so it is recommend you do not run this check with short intervals.

For MRTG output, returns the percent of free-space-map on the first line, and the number of pages currently used on the second line.

fsm_relations

(`symlink: check_postgres_fsm_relations`) Checks how close a cluster is to the Postgres **max_fsm_relations** setting. This action will only work for databases of 8.2 or higher, and it requires the contrib module **pg_freespacemap** be installed. The *–warning* and *–critical* options should be expressed as percentages. The number of used relations in the free-space-map is determined by looking in the pg_freespacemap_relations view. The default values are **85%** for the warning and **95%** for the critical.

Example 1: Give a warning when our cluster has used up 80% of the free-space relations, with pg_freespacemap installed in database dylan

```
check_postgres_fsm_relations -dbname=dylan -warning="75%"
```

While you need to pass in the name of the database where pg_freespacemap is installed, you only need to run this check once per cluster. Also, checking this information does require obtaining special locks on the free-space-map, so it is recommend you do not run this check with short intervals.

For MRTG output, returns the percent of free-space-map on the first line, the number of relations currently used on the second line.

hitratio

(`symlink: check_postgres_hitratio`) Checks the hit ratio of all databases and complains when they are too low. There is no need to run this command more than once per database cluster. Databases can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. They can also be filtered by the owner of the database with the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The warning and critical options should be specified as percentages. There are not defaults for this action: the warning and critical must be specified. The warning value cannot be greater than the critical value. The output returns all databases sorted by hitratio, smallest first.

Example: Warn if any database on host flagg is less than 90% in hitratio, and critical if less then 80%.

```
check_postgres_hitratio -host=flagg -warning='90%' -critical='80%'
```

For MRTG output, returns the percentage of the database with the smallest hitratio on the first line, and the name of the database on the fourth line.

hot_standby_delay

(`symlink: check_hot_standby_delay`) Checks the streaming replication lag by computing the delta between the current xlog position of a master server and the replay location of a slave connected to it. The slave server must be in hot_standby (e.g. read only) mode, therefore the minimum version to use this action is Postgres 9.0. The *–warning* and *–critical* options are the delta between the xlog locations. Since these values are byte offsets in the WAL they should match the expected transaction volume of your application to prevent false positives or negatives.

The first "–dbname", "–host", and "–port", etc. options are considered the master; the second belongs to the slave.

Byte values should be based on the volume of transactions needed to have the streaming replication disconnect from the master because of too much lag, determined by the Postgres configuration variable **wal_keep_segments** . For units of time, valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or abbreviated to just the first letter. When specifying both, in the form ' *bytes* and *time* ', both conditions must be true for the threshold to be met.

You must provide information on how to reach the databases by providing a comma separated list to the –dbhost and –dbport parameters, such as "–dbport=5432,5543". If not given, the action fails.

Example 1: Warn a database with a local replica on port 5433 is behind on any xlog replay at all

```
check_hot_standby_delay -dbport=5432,5433 -warning='1'
```

Example 2: Give a critical if the last transaction replica1 receives is more than 10 minutes ago

```
check_hot_standby_delay -dbhost=master,replica1 -critical='10 min'
```

Example 3: Allow replica1 to be 1 WAL segment behind, if the master is momentarily seeing more activity than the streaming replication connection can handle, or 10 minutes behind, if the master is seeing very little activity and not processing any transactions, but not both, which would indicate a lasting problem with the replication connection.

```
check_hot_standby_delay -dbhost=master,replica1 -warning='1048576 and 2 min' -critical='16
```

relation_size

index_size

table_size

indexes_size

total_relation_size

(symlinks: `check_postgres_relation_size` , `check_postgres_index_size` , `check_postgres_table_size` , `check_postgres_indexes_size` , and `check_postgres_total_relation_size`)

The actions **relation_size** and **index_size** check for a relation (table, index, materialized view), respectively an index that has grown too big, using the **pg_relation_size()** function.

The action **table_size** checks tables and materialized views using **pg_table_size()** , i.e. including relation forks and TOAST table.

The action **indexes_size** checks tables and materialized views for the size of the attached indexes using **pg_indexes_size()** .

The action **total_relation_size** checks relations using **pg_total_relation_size()** , i.e. including relation forks, indexes and TOAST table.

Relations can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. Relations can also be filtered by the user that owns them, by using the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The values for the *–warning* and *–critical* options are file sizes, and may have units of bytes, kilobytes, megabytes, gigabytes, terabytes, or exabytes. Each can be abbreviated to the first letter. If no units are given, bytes are assumed. There are no default values: both the warning and the critical option must be given. The return text shows the size of the largest relation found.

If the *–showperf* option is enabled, *all* of the relations with their sizes will be given. To prevent this, it is recommended that you set the *–perflimit* option, which will cause the query to do a `ORDER BY size DESC LIMIT (perflimit)` .

Example 1: Give a critical if any table is larger than 600MB on host burrick.

```
check_postgres_table_size -critical='600 MB' -warning='600 MB' -host=burrick
```

Example 2: Warn if the table products is over 4 GB in size, and give a critical at 4.5 GB.

```
check_postgres_table_size -host=burrick -warning='4 GB' -critical='4.5 GB' -include=produc
```

Example 3: Warn if any index not owned by postgres goes over 500 MB.

```
check_postgres_index_size -port=5432 -excludeuser=postgres -w 500MB -c 600MB
```

For MRTG output, returns the size in bytes of the largest relation, and the name of the database and relation as the fourth line.

last_analyze

last_vacuum

last_autoanalyze

last_autovacuum

(symlinks: `check_postgres_last_analyze` , `check_postgres_last_vacuum` , `check_postgres_last_autoanalyze` , and `check_postgres_last_autovacuum`) Checks how long it has been since vacuum (or analyze) was last run on each table in one or more databases. Use of these actions requires that the target database is version 8.3 or greater, or that the version is 8.2 and the configuration variable **stats_row_level** has been enabled. Tables can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. Tables can also be filtered by their owner by use of the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The units for *–warning* and *–critical* are specified as times. Valid units are seconds, minutes, hours, and days; all can be abbreviated to the first letter. If no units are given, 'seconds' are assumed. The default values are '1 day' and '2 days'. Please note that there are cases in which this field does not get automatically populated. If certain tables are giving you problems, make sure that they have dead rows to vacuum, or just exclude them from the test.

The schema named 'information_schema' is excluded from this test, as the only tables it contains are small and do not change.

Note that the non-'auto' versions will also check on the auto versions as well. In other words, using last_vacuum will report on the last vacuum, whether it was a normal vacuum, or one run by the autovacuum daemon.

Example 1: Warn if any table has not been vacuumed in 3 days, and give a critical at a week, for host wormwood

```
check_postgres_last_vacuum -host=wormwood -warning='3d' -critical='7d'
```

Example 2: Same as above, but skip tables belonging to the users 'eve' or 'mallory'

```
check_postgres_last_vacuum -host=wormwood -warning='3d' -critical='7d' -excludeuser=eve,ma
```

For MRTG output, returns (on the first line) the LEAST amount of time in seconds since a table was last vacuumed or analyzed. The fourth line returns the name of the database and name of the table.

listener

(`symlink: check_postgres_listener`) Confirm that someone is listening for one or more specific strings (using the LISTEN/NOTIFY system), by looking at the pg_listener table. Only one of warning or critical is needed. The format is a simple string representing the LISTEN target, or a tilde character followed by a string for a regular expression check. Note that this check will not work on versions of Postgres 9.0 or higher.

Example 1: Give a warning if nobody is listening for the string bucardo_mcp_ping on ports 5555 and 5556

```
check_postgres_listener -port=5555,5556 -warning=bucardo_mcp_ping
```

Example 2: Give a critical if there are no active LISTEN requests matching 'grimm' on database oskar

```
check_postgres_listener -db oskar -critical=~grimm
```

For MRTG output, returns a 1 or a 0 on the first, indicating success or failure. The name of the notice must be provided via the *–mrtg* option.

locks

(`symlink: check_postgres_locks`) Check the total number of locks on one or more databases. There is no need to run this more than once per database cluster. Databases can be filtered with the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details.

The *–warning* and *–critical* options can be specified as simple numbers, which represent the total number of locks, or they can be broken down by type of lock. Valid lock names are `'total'`, `'waiting'`, or the name of a lock type used by Postgres. These names are case-insensitive and do not need the "lock" part on the end, so **exclusive** will match 'ExclusiveLock'. The format is name=number, with different items separated by colons or semicolons (or any other symbol).

Example 1: Warn if the number of locks is 100 or more, and critical if 200 or more, on host garrett

```
check_postgres_locks -host=garrett -warning=100 -critical=200
```

Example 2: On the host artemus, warn if 200 or more locks exist, and give a critical if over 250 total locks exist, or if over 20 exclusive locks exist, or if over 5 connections are waiting for a lock.

```
check_postgres_locks -host=artemus -warning=200 -critical="total=250:waiting=5:exclusive=2
```

For MRTG output, returns the number of locks on the first line, and the name of the database on the fourth line.

logfile

(`symlink: check_postgreslogfile`) Ensures that the logfile is in the expected location and is being logged to. This action issues a command that throws an error on each database it is checking, and ensures that the message shows up in the logs. It scans the various log* settings inside of Postgres to figure out where the logs should be. If you are using syslog, it does a rough (but not foolproof) scan of */etc/syslog.conf* . Alternatively, you can provide the name of the logfile with the *–logfile* option. This is especially useful if the logs

have a custom rotation scheme driven be an external program. The –**logfile** option supports the following escape characters: `%Y %m %d %H` , which represent the current year, month, date, and hour respectively. An error is always reported as critical unless the warning option has been passed in as a non-zero value. Other than that specific usage, the `-warning` and `-critical` options should *not* be used.

Example 1: On port 5432, ensure the logfile is being written to the file /home/greg/pg8.2.log

```
check_postgres_logfile -port=5432 -logfile=/home/greg/pg8.2.log
```

Example 2: Same as above, but raise a warning, not a critical

```
check_postgres_logfile -port=5432 -logfile=/home/greg/pg8.2.log -w 1
```

For MRTG output, returns a 1 or 0 on the first line, indicating success or failure. In case of a failure, the fourth line will provide more detail on the failure encountered.

new_version_bc

(`symlink: check_postgres_new_version_bc`) Checks if a newer version of the Bucardo program is available. The current version is obtained by running `bucardo_ctl -version` . If a major upgrade is available, a warning is returned. If a revision upgrade is available, a critical is returned. (Bucardo is a master to slave, and master to master replication system for Postgres: see https://bucardo.org/ for more information). See also the information on the `-get_method` option.

new_version_box

(`symlink: check_postgres_new_version_box`) Checks if a newer version of the boxinfo program is available. The current version is obtained by running `boxinfo.pl -version` . If a major upgrade is available, a warning is returned. If a revision upgrade is available, a critical is returned. (boxinfo is a program for grabbing important information from a server and putting it into a HTML format: see https://bucardo.org/Boxinfo/ for more information). See also the information on the `-get_method` option.

new_version_cp

(`symlink: check_postgres_new_version_cp`) Checks if a newer version of this program (check_postgres.pl) is available, by grabbing the version from a small text file on the main page of the home page for the project. Returns a warning if the returned version does not match the one you are running. Recommended interval to check is once a day. See also the information on the `-get_method` option.

new_version_pg

(`symlink: check_postgres_new_version_pg`) Checks if a newer revision of Postgres exists for each database connected to. Note that this only checks for revision, e.g. going from 8.3.6 to 8.3.7. Revisions are always 100% binary compatible and involve no dump and restore to upgrade. Revisions are made to address bugs, so upgrading as soon as possible is always recommended. Returns a warning if you do not have the latest revision. It is recommended this check is run at least once a day. See also the information on the `-get_method` option.

new_version_tnm

(`symlink: check_postgres_new_version_tnm`) Checks if a newer version of the tail_n_mail program is available. The current version is obtained by running `tail_n_mail -version` . If a major upgrade is available, a warning is returned. If a revision upgrade is available, a critical is returned. (tail_n_mail is a log monitoring tool that can send mail when interesting events appear in your Postgres logs. See: https://bucardo.org/tail_n_mail/ for more information). See also the information on the `-get_method` option.

pgb_pool_cl_active

pgb_pool_cl_waiting

pgb_pool_sv_active

pgb_pool_sv_idle

pgb_pool_sv_used

pgb_pool_sv_tested

pgb_pool_sv_login

pgb_pool_maxwait

(symlinks: `check_postgres_pgb_pool_cl_active` , `check_postgres_pgb_pool_cl_waiting` , `check_postgres_pgb_pool_sv_active` , `check_postgres_pgb_pool_sv_idle` , `check_postgres_pgb_pool_sv_used` , `check_postgres_pgb_pool_sv_tested` , `check_postgres_pgb_pool_sv_login` , and `check_postgres_pgb_pool_maxwait`)

Examines pgbouncer's pool statistics. Each pool has a set of "client" connections, referring to connections from external clients, and "server" connections, referring to connections to PostgreSQL itself. The related check*postgres actions are prefixed by "cl"* and "sv_", respectively. Active client connections are those connections currently linked with an active server connection. Client connections may also be "waiting", meaning they have not yet been allocated a server connection. Server connections are "active" (linked to a client), "idle" (standing

by for a client connection to link with), "used" (just unlinked from a client, and not yet returned to the idle pool), "tested" (currently being tested) and "login" (in the process of logging in). The maxwait value shows how long in seconds the oldest waiting client connection has been waiting.

pgbouncer_backends

(`symlink: check_postgres_pgbouncer_backends`) Checks the current number of connections for one or more databases through pgbouncer, and optionally compares it to the maximum allowed, which is determined by the pgbouncer configuration variable **max_client_conn** . The *–warning* and *–critical* options can take one of three forms. First, a simple number can be given, which represents the number of connections at which the alert will be given. This choice does not use the **max_connections** setting. Second, the percentage of available connections can be given. Third, a negative number can be given which represents the number of connections left until **max_connections** is reached. The default values for *–warning* and *–critical* are '90%' and '95%'. You can also filter the databases by use of the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details.

To view only non-idle processes, you can use the *–noidle* argument. Note that the user you are running as (either connecting directly or switching via *–role*) must be a superuser for this to work properly.

Example 1: Give a warning when the number of connections on host quirm reaches 120, and a critical if it reaches 150.

```
check_postgres_pgbouncer_backends -host=quirm -warning=120 -critical=150 -p 6432 -u pgbou
```

Example 2: Give a critical when we reach 75% of our max_connections setting on hosts lancre or lancre2.

```
check_postgres_pgbouncer_backends -warning='75%' -critical='75%' -host=lancre,lancre2 -p 6
```

Example 3: Give a warning when there are only 10 more connection slots left on host plasmid, and a critical when we have only 5 left.

```
check_postgres_pgbouncer_backends -warning=-10 -critical=-5 -host=plasmid -p 6432 -u pgbou
```

For MRTG output, the number of connections is reported on the first line, and the fourth line gives the name of the database, plus the current max_client_conn. If more than one database has been queried, the one with the highest number of connections is output.

pgbouncer_checksum

(`symlink: check_postgres_pgbouncer_checksum`) Checks that all the pgBouncer settings are the same as last time you checked. This is done by generating a checksum of a sorted list of setting names and their values. Note that you shouldn't specify the database name, it will automatically default to

pgbouncer. Either the *–warning* or the *–critical* option should be given, but not both. The value of each one is the checksum, a 32-character hexadecimal value. You can run with the special `-critical=0` option to find out an existing checksum.

This action requires the Digest::MD5 module.

Example 1: Find the initial checksum for pgbouncer configuration on port 6432 using the default user (usually postgres)

```
check_postgres_pgbouncer_checksum -port=6432 -critical=0
```

Example 2: Make sure no settings have changed and warn if so, using the checksum from above.

```
check_postgres_pgbouncer_checksum -port=6432 -warning=cd2f3b5e129dc2b4f5c0f6d8d2e64231
```

For MRTG output, returns a 1 or 0 indicating success of failure of the checksum to match. A checksum must be provided as the `-mrtg` argument. The fourth line always gives the current checksum.

pgbouncer_maxwait

(`symlink: check_postgres_pgbouncer_maxwait`) Checks how long the first (oldest) client in the queue has been waiting, in seconds. If this starts increasing, then the current pool of servers does not handle requests quick enough. Reason may be either overloaded server or just too small of a pool_size setting in pbouncer config file. Databases can be filtered by use of the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. The values or the *–warning* and *–critical* options are units of time, and must be provided (no default). Valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or abbreviated to just the first letter. If no units are given, the units are assumed to be seconds.

This action requires Postgres 8.3 or better.

Example 1: Give a critical if any transaction has been open for more than 10 minutes:

```
check_postgres_pgbouncer_maxwait -p 6432 -u pgbouncer -critical='10 minutes'
```

For MRTG output, returns the maximum time in seconds a transaction has been open on the first line. The fourth line gives the name of the database.

pgagent_jobs

(`symlink: check_postgres_pgagent_jobs`) Checks that all the pgAgent jobs that have executed in the preceding interval of time have succeeded. This is done by checking for any steps that have a non-zero result.

Either `-warning` or `-critical` , or both, may be specified as times, and jobs will be checked for failures withing the specified periods of time before

the current time. Valid units are seconds, minutes, hours, and days; all can be abbreviated to the first letter. If no units are given, 'seconds' are assumed.

Example 1: Give a critical when any jobs executed in the last day have failed.

```
check_postgres_pgagent_jobs -critical=1d
```

Example 2: Give a warning when any jobs executed in the last week have failed.

```
check_postgres_pgagent_jobs -warning=7d
```

Example 3: Give a critical for jobs that have failed in the last 2 hours and a warning for jobs that have failed in the last 4 hours:

```
check_postgres_pgagent_jobs -critical=2h -warning=4h
```

prepared_txns

(`symlink: check_postgres_prepared_txns`) Check on the age of any existing prepared transactions. Note that most people will NOT use prepared transactions, as they are part of two-part commit and complicated to maintain. They should also not be confused with prepared STATEMENTS, which is what most people think of when they hear prepare. The default value for a warning is 1 second, to detect any use of prepared transactions, which is probably a mistake on most systems. Warning and critical are the number of seconds a prepared transaction has been open before an alert is given.

Example 1: Give a warning on detecting any prepared transactions:

```
check_postgres_prepared_txns -w 0
```

Example 2: Give a critical if any prepared transaction has been open longer than 10 seconds, but allow up to 360 seconds for the database 'shrike':

```
check_postgres_prepared_txns -critical=10 -exclude=shrike
check_postgres_prepared_txns -critical=360 -include=shrike
```

For MRTG output, returns the number of seconds the oldest transaction has been open as the first line, and which database is came from as the final line.

query_runtime

(`symlink: check_postgres_query_runtime`) Checks how long a specific query takes to run, by executing a "EXPLAIN ANALYZE" against it. The *–warning* and *–critical* options are the maximum amount of time the query should take. Valid units are seconds, minutes, and hours; any can be abbreviated to the first letter. If no units are given, 'seconds' are assumed. Both the warning and the critical option must be given. The name of the view or function to be run must be passed in to the *–queryname* option. It must consist of a single word (or schema.word), with optional parens at the end.

Example 1: Give a critical if the function named "speedtest" fails to run in 10 seconds or less.

```
check_postgres_query_runtime -queryname='speedtest()' -critical=10 -warning=10
```

For MRTG output, reports the time in seconds for the query to complete on the first line. The fourth line lists the database.

query_time

(`symlink: check_postgres_query_time`) Checks the length of running queries on one or more databases. There is no need to run this more than once on the same database cluster. Note that this already excludes queries that are "idle in transaction". Databases can be filtered by using the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. You can also filter on the user running the query with the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The values for the *–warning* and *–critical* options are amounts of time, and at least one must be provided (no defaults). Valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or abbreviated to just the first letter. If no units are given, the unit is assumed to be seconds.

This action requires Postgres 8.1 or better.

Example 1: Give a warning if any query has been running longer than 3 minutes, and a critical if longer than 5 minutes.

```
check_postgres_query_time -port=5432 -warning='3 minutes' -critical='5 minutes'
```

Example 2: Using default values (2 and 5 minutes), check all databases except those starting with 'template'.

```
check_postgres_query_time -port=5432 -exclude=~^template
```

Example 3: Warn if user 'don' has a query running over 20 seconds

```
check_postgres_query_time -port=5432 -includeuser=don -warning=20s
```

For MRTG output, returns the length in seconds of the longest running query on the first line. The fourth line gives the name of the database.

replicate_row

(`symlink: check_postgres_replicate_row`) Checks that master-slave replication is working to one or more slaves.

The first "–dbname", "–host", and "–port", etc. options are considered the master; subsequent uses are the slaves. The values or the *–warning* and *–critical* options are units of time, and at least one must be provided (no defaults). Valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or

abbreviated to just the first letter. If no units are given, the units are assumed to be seconds.

This check updates a single row on the master, and then measures how long it takes to be applied to the slaves. To do this, you need to pick a table that is being replicated, then find a row that can be changed, and is not going to be changed by any other process. A specific column of this row will be changed from one value to another. All of this is fed to the `repinfo` option, and should contain the following options, separated by commas: table name, primary key, key id, column, first value, second value.

Example 1: Slony is replicating a table named 'orders' from host 'alpha' to host 'beta', in the database 'sales'. The primary key of the table is named id, and we are going to test the row with an id of 3 (which is historical and never changed). There is a column named 'salesrep' that we are going to toggle from a value of 'slon' to 'nols' to check on the replication. We want to throw a warning if the replication does not happen within 10 seconds.

```
check_postgres_replicate_row -host=alpha -dbname=sales -host=beta
-dbname=sales -warning=10 -repinfo=orders,id,3,salesrep,slon,nols
```

Example 2: Bucardo is replicating a table named 'receipt' from host 'green' to hosts 'red', 'blue', and 'yellow'. The database for both sides is 'public'. The slave databases are running on port 5455. The primary key is named 'receipt_id', the row we want to use has a value of 9, and the column we want to change for the test is called 'zone'. We'll toggle between 'north' and 'south' for the value of this column, and throw a critical if the change is not on all three slaves within 5 seconds.

```
check_postgres_replicate_row -host=green -port=5455 -host=red,blue,yellow
-critical=5 -repinfo=receipt,receipt_id,9,zone,north,south
```

For MRTG output, returns on the first line the time in seconds the replication takes to finish. The maximum time is set to 4 minutes 30 seconds: if no replication has taken place in that long a time, an error is thrown.

replication_slots

(`symlink: check_postgres_replication_slots`) Check the quantity of WAL retained for any replication slots in the target database cluster. This is handy for monitoring environments where all WAL archiving and replication is taking place over replication slots.

Warning and critical are total bytes retained for the slot. E.g:

```
check_postgres_replication_slots -port=5432 -host=yellow -warning=32M -critical=64M
```

Specific named slots can be monitored using –include/–exclude

same_schema

(`symlink: check_postgres_same_schema`) Verifies that two or more databases are identical as far as their schema (but not the data within). Unlike most other actions, this has no warning or critical criteria - the databases are either in sync, or are not. If they are different, a detailed list of the differences is presented.

You may want to exclude or filter out certain differences. The way to do this is to add strings to the `-filter` option. To exclude a type of object, use "noname", where 'name' is the type of object, for example, "noschema". To exclude objects of a certain type by a regular expression against their name, use "noname=regex". See the examples below for a better understanding.

The types of objects that can be filtered include:

user
schema
table
view
index
sequence
constraint
trigger
function

The filter option "noposition" prevents verification of the position of columns within a table.

The filter option "nofuncbody" prevents comparison of the bodies of all functions.

The filter option "noperm" prevents comparison of object permissions.

To provide the second database, just append the differences to the first one by a call to the appropriate connection argument. For example, to compare databases on hosts alpha and bravo, use "–dbhost=alpha,bravo". Also see the examples below.

If only a single host is given, it is assumed we are doing a "time-based" report. The first time this is run a snapshot of all the items in the database is saved to a local file. When you run it again, that snapshot is read in and becomes "database #2" and is compared to the current database.

To replace the old stored file with the new version, use the –replace argument.

If you need to write the stored file to a specific directory, use the –audit-file-dir argument.

To avoid false positives on value based checks caused by replication lag on asynchronous replicas, use the *–assume-async* option.

To enable snapshots at various points in time, you can use the "–suffix" argument to make the filenames unique to each run. See the examples below.

Example 1: Verify that two databases on hosts star and line are the same:

```
check_postgres_same_schema -dbhost=star,line
```

Example 2: Same as before, but exclude any triggers with "slony" in their name

```
check_postgres_same_schema -dbhost=star,line -filter="notrigger=slony"
```

Example 3: Same as before, but also exclude all indexes

```
check_postgres_same_schema -dbhost=star,line -filter="notrigger=slony noindexes"
```

Example 4: Check differences for the database "battlestar" on different ports

```
check_postgres_same_schema -dbname=battlestar -dbport=5432,5544
```

Example 5: Create a daily and weekly snapshot file

```
check_postgres_same_schema -dbname=cylon -suffix=daily
check_postgres_same_schema -dbname=cylon -suffix=weekly
```

Example 6: Run a historical comparison, then replace the file

```
check_postgres_same_schema -dbname=cylon -suffix=daily -replace
```

Example 7: Verify that two databases on hosts star and line are the same, excluding value data (i.e. sequence last_val):

```
check_postgres_same_schema -dbhost=star,line -assume-async
```

sequence

(`symlink: check_postgres_sequence`) Checks how much room is left on all sequences in the database. This is measured as the percent of total possible values that have been used for each sequence. The *–warning* and *–critical* options should be expressed as percentages. The default values are **85%** for the warning and **95%** for the critical. You may use –include and –exclude to control which sequences are to be checked. Note that this check does account for unusual **minvalue** and **increment by** values. By default it does not care if the sequence is set to cycle or not, and by passing *–skipcycled* sequenced set to cycle are reported with 0% usage.

The output for Nagios gives the name of the sequence, the percentage used, and the number of 'calls' left, indicating how many more times nextval can be called on that sequence before running into the maximum value.

The output for MRTG returns the highest percentage across all sequences on the first line, and the name of each sequence with that percentage on the fourth line, separated by a "|" (pipe) if there are more than one sequence at that percentage.

Example 1: Give a warning if any sequences are approaching 95% full.

```
check_postgres_sequence -dbport=5432 -warning=95%
```

Example 2: Check that the sequence named "orders_id_seq" is not more than half full.

```
check_postgres_sequence -dbport=5432 -critical=50% -include=orders_id_seq
```

settings_checksum

(`symlink: check_postgres_settings_checksum`) Checks that all the Postgres settings are the same as last time you checked. This is done by generating a checksum of a sorted list of setting names and their values. Note that different users in the same database may have different checksums, due to ALTER USER usage, and due to the fact that superusers see more settings than ordinary users. Either the *–warning* or the *–critical* option should be given, but not both. The value of each one is the checksum, a 32-character hexadecimal value. You can run with the special `-critical=0` option to find out an existing checksum.

This action requires the Digest::MD5 module.

Example 1: Find the initial checksum for the database on port 5555 using the default user (usually postgres)

```
check_postgres_settings_checksum -port=5555 -critical=0
```

Example 2: Make sure no settings have changed and warn if so, using the checksum from above.

```
check_postgres_settings_checksum -port=5555 -warning=cd2f3b5e129dc2b4f5c0f6d8d2e64231
```

For MRTG output, returns a 1 or 0 indicating success of failure of the checksum to match. A checksum must be provided as the `-mrtg` argument. The fourth line always gives the current checksum.

slony_status

(`symlink: check_postgres_slony_status`) Checks in the status of a Slony cluster by looking at the results of Slony's sl_status view. This is returned as the number of seconds of "lag time". The *–warning* and *–critical* options should be expressed as times. The default values are **60 seconds** for the warning and **300 seconds** for the critical.

The optional argument *–schema* indicated the schema that Slony is installed under. If it is not given, the schema will be determined automatically each time this check is run.

Example 1: Give a warning if any Slony is lagged by more than 20 seconds

```
check_postgres_slony_status -warning 20
```

Example 2: Give a critical if Slony, installed under the schema "_slony", is over 10 minutes lagged

```
check_postgres_slony_status -schema=_slony -critical=600
```

timesync

(`symlink: check_postgres_timesync`) Compares the local system time with the time reported by one or more databases. The *–warning* and *–critical* options represent the number of seconds between the two systems before an alert is given. If neither is specified, the default values are used, which are '2' and '5'. The warning value cannot be greater than the critical value. Due to the non-exact nature of this test, values of '0' or '1' are not recommended.

The string returned shows the time difference as well as the time on each side written out.

Example 1: Check that databases on hosts ankh, morpork, and klatch are no more than 3 seconds off from the local time:

```
check_postgres_timesync -host=ankh,morpork,klatch -critical=3
```

For MRTG output, returns one the first line the number of seconds difference between the local time and the database time. The fourth line returns the name of the database.

txn_idle

(`symlink: check_postgres_txn_idle`) Checks the number and duration of "idle in transaction" queries on one or more databases. There is no need to run this more than once on the same database cluster. Databases can be filtered by using the *–include* and *–exclude* options. See the "BASIC FILTERING" section below for more details.

The *–warning* and *–critical* options are given as units of time, signed integers, or integers for units of time, and at least one must be provided (there are no defaults). Valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or abbreviated to just the first letter. If no units are given and the numbers are unsigned, the units are assumed to be seconds.

This action requires Postgres 8.3 or better.

As of PostgreSQL 10, you can just GRANT *pg_read_all_stats* to an unprivileged user account. In all earlier versions, superuser privileges are required to see the queries of all users in the system; UNKNOWN is returned if queries cannot be checked. To only include queries by the connecting user, use *–includeuser* .

Example 1: Give a warning if any connection has been idle in transaction for more than 15 seconds:

```
check_postgres_txn_idle -port=5432 -warning='15 seconds'
```

Example 2: Give a warning if there are 50 or more transactions

```
check_postgres_txn_idle -port=5432 -warning='+50'
```

Example 3: Give a critical if 5 or more connections have been idle in transaction for more than 10 seconds:

```
check_postgres_txn_idle -port=5432 -critical='5 for 10 seconds'
```

For MRTG output, returns the time in seconds the longest idle transaction has been running. The fourth line returns the name of the database and other information about the longest transaction.

txn_time

(`symlink: check_postgres_txn_time`) Checks the length of open transactions on one or more databases. There is no need to run this command more than once per database cluster. Databases can be filtered by use of the *–include* and *–exclude* options. See the "BASIC FILTERING" section for more details. The owner of the transaction can also be filtered, by use of the *–includeuser* and *–excludeuser* options. See the "USER NAME FILTERING" section for more details.

The values or the *–warning* and *–critical* options are units of time, and at least one must be provided (no default). Valid units are 'seconds', 'minutes', 'hours', or 'days'. Each may be written singular or abbreviated to just the first letter. If no units are given, the units are assumed to be seconds.

This action requires Postgres 8.3 or better.

Example 1: Give a critical if any transaction has been open for more than 10 minutes:

```
check_postgres_txn_time -port=5432 -critical='10 minutes'
```

Example 1: Warn if user 'warehouse' has a transaction open over 30 seconds

```
check_postgres_txn_time -port-5432 -warning=30s -includeuser=warehouse
```

For MRTG output, returns the maximum time in seconds a transaction has been open on the first line. The fourth line gives the name of the database.

txn_wraparound

(`symlink: check_postgres_txn_wraparound`) Checks how close to transaction wraparound one or more databases are getting. The *–warning* and *–critical* options indicate the number of transactions done, and must be a positive integer. If either option is not given, the default values of 1.3 and 1.4 billion are used. There is no need to run this command more than once per database cluster. For a more detailed discussion of what this number represents and what to do about it, please

visit the page https://www.postgresql.org/docs/current/static/routine-vacuuming.html#VACUUM-FOR-WRAPAROUND

The warning and critical values can have underscores in the number for legibility, as Perl does.

Example 1: Check the default values for the localhost database

```
check_postgres_txn_wraparound -host=localhost
```

Example 2: Check port 6000 and give a critical when 1.7 billion transactions are hit:

```
check_postgres_txn_wraparound -port=6000 -critical=1_700_000_000
```

For MRTG output, returns the highest number of transactions for all databases on line one, while line 4 indicates which database it is.

version

(`symlink: check_postgres_version`) Checks that the required version of Postgres is running. The *–warning* and *–critical* options (only one is required) must be of the format **X.Y** or **X.Y.Z** where **X** is the major version number, **Y** is the minor version number, and **Z** is the revision.

Example 1: Give a warning if the database on port 5678 is not version 8.4.10:

```
check_postgres_version -port=5678 -w=8.4.10
```

Example 2: Give a warning if any databases on hosts valley,grain, or sunshine is not 8.3:

```
check_postgres_version -H valley,grain,sunshine -critical=8.3
```

For MRTG output, reports a 1 or a 0 indicating success or failure on the first line. The fourth line indicates the current version. The version must be provided via the `-mrtg` option.

wal_files

(`symlink: check_postgres_wal_files`) Checks how many WAL files exist in the *pg_xlog* directory (PostgreSQL 10 and later" *pg_wal*), which is found off of your **data_directory** , sometimes as a symlink to another physical disk for performance reasons. If the *–lsfunc* option is not used then this action must be run as a superuser, in order to access the contents of the *pg_xlog* directory. The minimum version to use this action is Postgres 8.1. The *–warning* and *–critical* options are simply the number of files in the *pg_xlog* directory. What number to set this to will vary, but a general guideline is to put a number slightly higher than what is normally there, to catch problems early.

Normally, WAL files are closed and then re-used, but a long-running open transaction, or a faulty **archive_command** script, may cause Postgres to

create too many files. Ultimately, this will cause the disk they are on to run out of space, at which point Postgres will shut down.

To avoid connecting as a database superuser, a wrapper function around `pg_ls_dir()` should be defined as a superuser with SECURITY DEFINER, and the *–lsfunc* option used. This example function, if defined by a superuser, will allow the script to connect as a normal user *nagios* with *–lsfunc=ls_xlog_dir*

```
BEGIN;
CREATE FUNCTION ls_xlog_dir()
    RETURNS SETOF TEXT
    AS $$ SELECT pg_ls_dir('pg_xlog') $$
    LANGUAGE SQL
    SECURITY DEFINER;
REVOKE ALL ON FUNCTION ls_xlog_dir() FROM PUBLIC;
GRANT EXECUTE ON FUNCTION ls_xlog_dir() to nagios;
COMMIT;
```

Example 1: Check that the number of ready WAL files is 10 or less on host "pluto", using a wrapper function `ls_xlog_dir` to avoid the need for superuser permissions

```
check_postgres_archive_ready -host=pluto -critical=10 -lsfunc=ls_xlog_dir
```

For MRTG output, reports the number of WAL files on line 1.

rebuild_symlinks

rebuild_symlinks_force

This action requires no other arguments, and does not connect to any databases, but simply creates symlinks in the current directory for each action, in the form **check*postgres*** . If the file already exists, it will not be overwritten. If the action is rebuild_symlinks_force, then symlinks will be overwritten. The option –symlinks is a shorter way of saying –action=rebuild_symlinks

BASIC FILTERING

The options *–include* and *–exclude* can be combined to limit which things are checked, depending on the action. The name of the database can be filtered when using the following actions: backends, database_size, locks, query_time, txn_idle, and txn_time. The name of a relation can be filtered when using the following actions: bloat, index_size, table_size, relation_size, last_vacuum, last_autovacuum, last_analyze, and last_autoanalyze. The name of a setting can be filtered when using the settings_checksum action. The name of a file system can be filtered when using the disk_space action.

If only an include option is given, then ONLY those entries that match will be checked. However, if given both exclude and include, the exclusion is done first,

and the inclusion after, to reinstate things that may have been excluded. Both *–include* and *–exclude* can be given multiple times, and/or as comma-separated lists. A leading tilde will match the following word as a regular expression.

To match a schema, end the search term with a single period. Leading tildes can be used for schemas as well.

Be careful when using filtering: an inclusion rule on the backends, for example, may report no problems not only because the matching database had no backends, but because you misspelled the name of the database!

Examples:

Only checks items named pg_class:

```
-include=pgclass
```

Only checks items containing the letters 'pg':

```
-include=~pg
```

Only check items beginning with 'pg':

```
-include=~^pg_
```

Exclude the item named 'test':

```
-exclude=test
```

Exclude all items containing the letters 'test:

```
-exclude=~test
```

Exclude all items in the schema 'pg_catalog':

```
-exclude='pg_catalog.'
```

Exclude all items in the 'pg_temp_nnn' per-session temporary schemas:

```
-exclude=~^pgtemp.
```

Exclude all items containing the letters 'ace', but allow the item 'faceoff':

```
-exclude=~ace -include=faceoff
```

Exclude all items which start with the letters 'pg_', which contain the letters 'slon', or which are named 'sql_settings' or 'green'. Specifically check items with the letters 'prod' in their names, and always check the item named 'pg

relname':

```
-exclude=~^pg,~slon,sql_settings -exclude=green -include=~prod,pg_relname
```

USER NAME FILTERING

The options *–includeuser* and *–excludeuser* can be used on some actions to only examine database objects owned by (or not owned by) one or more users. An *–includeuser* option always trumps an *–excludeuser* option. You can give each option more than once for multiple users, or you can give a comma-separated list. The actions that currently use these options are:

database_size
last_analyze
last_autoanalyze
last_vacuum
last_autovacuum
query_time
relation_size
txn_time

Examples:

Only check items owned by the user named greg:

```
-includeuser=greg
```

Only check items owned by either watson or crick:

```
-includeuser=watson,crick
```

Only check items owned by crick,franklin, watson, or wilkins:

```
-includeuser=watson -includeuser=franklin -includeuser=crick,wilkins
```

Check all items except for those belonging to the user scott:

```
-excludeuser=scott
```

TEST MODE

To help in setting things up, this program can be run in a "test mode" by specifying the *–test* option. This will perform some basic tests to make sure that the databases can be contacted, and that certain per-action prerequisites are met, such as whether the user is a superuser, if the version of Postgres is new enough, and if stats_row_level is enabled.

FILES

In addition to command-line configurations, you can put any options inside of a file. The file *.check_postgresrc* in the current directory will be used if found. If not found, then the file *~/.check_postgresrc* will be used. Finally, the file */etc/check_postgresrc* will be used if available. The format of the file is option = value, one per line. Any line starting with a '#' will be skipped. Any values loaded from a check_postgresrc file will be overwritten by command-line options. All

check_postgresrc files can be ignored by supplying a `-no-checkpostgresrc` argument.

ENVIRONMENT VARIABLES

The environment variable *$ENV{HOME}* is used to look for a *.check_postgresrc* file. The environment variable *$ENV{PGBINDIR}* is used to look for PostgreSQL binaries.

TIPS AND TRICKS

Since this program uses the **psql** program, make sure it is accessible to the user running the script. If run as a cronjob, this often means modifying the **PATH** environment variable.

If you are using Nagios in embedded Perl mode, use the `-action` argument instead of symlinks, so that the plugin only gets compiled one time.

DEPENDENCIES

Access to a working version of psql, and the following very standard Perl modules:

Cwd
Getopt::Long
File::Basename
File::Temp
Time::HiRes (if `$opt{showtime}` is set to true, which is the default)

The "settings_checksum" action requires the **Digest::MD5** module.

The "checkpoint" action requires the **Date::Parse** module.

Some actions require access to external programs. If psql is not explicitly specified, the command **`which`** is used to find it. The program **`/bin/df`** is needed by the "disk_space" action.

DEVELOPMENT

Development happens using the git system. You can clone the latest version by doing:

```
https://github.com/bucardo/check_postgres
git clone git://bucardo.org/check_postgres.git
```

MAILING LIST

Three mailing lists are available. For discussions about the program, bug reports, feature requests, and commit notices, send email to check_postgres@bucardo.org

https://mail.endcrypt.com/mailman/listinfo/check_postgres

A low-volume list for announcement of new versions and important notices is the 'check_postgres-announce' list:

https://mail.endcrypt.com/mailman/listinfo/check_postgres-announce

Source code changes (via git-commit) are sent to the 'check_postgres-commit' list:

https://mail.endcrypt.com/mailman/listinfo/check_postgres-commit

HISTORY

Items not specifically attributed are by GSM (Greg Sabino Mullane).

Version 2.26.0 Released April 3, 2023

```
    Add new action "pgbouncer_maxwait" (Ruslan Kabalin) [Github pull #59]

  For the bloat check, add option to populate all known databases,
      as well as includsion and exclusion regexes. (Giles Westwood) [Github pull #86]

  Add Partman premake check (Jens Wilke) [Github pull #196]

  Add -role flag to explicitly set the role of the user after connecting (David Christens

  Fix check_replication_slots on recently promoted servers (Christoph Berg)

  Allow the check_disk_space action to handle relative log_directory paths (jacksonfoz) [

  Fix MINPAGES and MINIPAGES in the "check_bloat" action (Christoph Moench-Tegeder) [Gith

  Replace 'which' with 'command -v' (Christoph Berg)

  Fix check_replication_slots on recently promoted servers (Christoph Berg)

  Fix undefined variable warning (Michael van Bracht) [Github pull #158]

  In the tests, force log_destination to stderr (Christoph Moench-Tegeder) [Github pull #

  Add to docs how to exclude all items in the 'pg_temp_nnn' per-session temporary schemas

  Various fixes for the CI system (Emre Hasegeli) [Github pull #181]

  Various improvements to the tests (Christoph Berg, Emre Hasegeli)
```

Version 2.25.0 Released February 3, 2020

```
    Allow same_schema objects to be included or excluded with -object and -skipobject
```

(Greg Sabino Mullane)

Fix to allow mixing service names and other connection parameters for same_schema
(Greg Sabino Mullane)

Version 2.24.0 Released May 30, 2018

Support new_version_pg for PG10
(Michael Pirogov)

Option to skip CYCLE sequences in action sequence
(Christoph Moench-Tegeder)

Output per-database perfdata for pgbouncer pool checks
(George Hansper)

German message translations
(Holger Jacobs)

Consider only client backends in query_time and friends
(David Christensen)

Version 2.23.0 Released October 31, 2017

Support PostgreSQL 10.
(David Christensen, Christoph Berg)

Change table_size to use pg_table_size() on 9.0+, i.e. include the TOAST table size in the numbers reported. Add new actions indexes_size and total_relation_size, using the respective pg_indexes_size() and pg_total_relation_size() functions. All size checks will now also check materialized views where applicable.
(Christoph Berg)

Connection errors are now always critical, not unknown.
(Christoph Berg)

New action replication_slots checking if logical or physical replication slots have accumulated too much data
(Glyn Astill)

Multiple same_schema improvements
(Glyn Astill)

Add Spanish message translations
(Luis Vazquez)

Allow a wrapper function to run wal_files and archive_ready actions as

```
  non-superuser
    (Joshua Elsasser)

Add some defensive casting to the bloat query
    (Greg Sabino Mullane)

Invoke psql with option -X
    (Peter Eisentraut)

Update postgresql.org URLs to use https.
    (Magnus Hagander)

check_txn_idle: Don't fail when query contains 'disabled' word
    (Marco Nenciarini)

check_txn_idle: Use state_change instead of query_start.
    (Sebastian Webber)

check_hot_standby_delay: Correct extra space in perfdata
    (Adrien Nayrat)

Remove \r from psql output as it can confuse some regexes
    (Greg Sabino Mullane)

Sort failed jobs in check_pgagent_jobs for stable output.
    (Christoph Berg)
```

Version 2.22.0 June 30, 2015

```
  Add xact timestamp support to hot_standby_delay.
  Allow the hot_standby_delay check to accept xlog byte position or
  timestamp lag intervals as thresholds, or even both at the same time.
    (Josh Williams)

Query all sequences per DB in parallel for action=sequence.
    (Christoph Berg)

Fix bloat check to use correct SQL depending on the server version.
    (Adrian Vondendriesch)

Show actual long-running query in query_time output
    (Peter Eisentraut)

Add explicit ORDER BY to the slony_status check to get the most lagged server.
    (Jeff Frost)

Improved multi-slave support in replicate_row.
```

(Andrew Yochum)

Change the way tables are quoted in replicate_row.
(Glyn Astill)

Don't swallow space before the -c flag when reporting errors
(Jeff Janes)

Fix and extend hot_standby_delay documentation
(Michael Renner)

Declare POD encoding to be utf8.
(Christoph Berg)

Version 2.21.0 September 24, 2013

Fix issue with SQL steps in check_pgagent_jobs for sql steps which perform deletes
(Rob Emery via github pull)

Install man page in section 1.
(Peter Eisentraut, bug 53, github issue 26)

Order lock types in check_locks output to make the ordering predictable;
setting SKIP_NETWORK_TESTS will skip the new_version tests; other minor test
suite fixes.
(Christoph Berg)

Fix same_schema check on 9.3 by ignoring relminmxid differences in pg_class
(Christoph Berg)

Version 2.20.1 June 24, 2013

Make connection check failures return CRITICAL not UNKNOWN
(Dominic Hargreaves)

Fix -reverse option when using string comparisons in custom queries
(Nathaniel Waisbrot)

Compute correct 'totalwastedbytes' in the bloat query
(Michael Renner)

Do not use pg_stats "inherited" column in bloat query, if the
database is 8.4 or older. (Greg Sabino Mullane, per bug 121)

Remove host reordering in hot_standby_delay check
(Josh Williams, with help from Jacobo Blasco)

Better output for the "simple" flag

(Greg Sabino Mullane)

Force same_schema to ignore the 'relallvisible' column
(Greg Sabino Mullane)

Version 2.20.0 March 13, 2013

Add check for pgagent jobs (David E. Wheeler)

Force STDOUT to use utf8 for proper output
(Greg Sabino Mullane; reported by Emmanuel Lesouef)

Fixes for Postgres 9.2: new pg_stat_activity view,
and use pg_tablespace_location, (Josh Williams)

Allow for spaces in item lists when doing same_schema.

Allow txn_idle to work again for < 8.3 servers by switching to query_time.

Fix the check_bloat SQL to take inherited tables into account,
and assume 2k for non-analyzed columns. (Geert Pante)

Cache sequence information to speed up same_schema runs.

Fix -excludeuser in check_txn_idle (Mika Eloranta)

Fix user clause handling in check_txn_idle (Michael van Bracht)

Adjust docs to show colon as a better separator inside args for locks
(Charles Sprickman)

Fix undefined $SQL2 error in check_txn_idle github issue 16

Prevent "uninitialized value" warnings when showing the port (Henrik Ahlgren)

Do not assume everyone has a HOME [github issue 23]

Version 2.19.0 January 17, 2012

Add the -assume-prod option (Cédric Villemain)

Add the cluster_id check (Cédric Villemain)

Improve settings_checksum and checkpoint tests (Cédric Villemain)

Do not do an inner join to pg_user when checking database size
(Greg Sabino Mullane; reported by Emmanuel Lesouef)

```
Use the full path when getting sequence information for same_schema.
    (Greg Sabino Mullane; reported by Cindy Wise)

Fix the formula for calculating xlog positions (Euler Taveira de Oliveira)

Better ordering of output for bloat check - make indexes as important
    as tables (Greg Sabino Mullane; reported by Jens Wilke)

Show the dbservice if it was used at top of same_schema output
    (Mike Blackwell)

Better installation paths (Greg Sabino Mullane, per bug 53)
```

Version 2.18.0 October 2, 2011

```
  Redo the same_schema action. Use new -filter argument for all filtering.
  Allow comparisons between any number of databases.
  Remove the dbname2, dbport2, etc. arguments.
  Allow comparison of the same db over time.

Swap db1 and db2 if the slave is 1 for the hot standby check (David E. Wheeler)

Allow multiple -schema arguments for the slony_status action (GSM and Jehan-Guillaume d

Fix ORDER BY in the last vacuum/analyze action (Nicolas Thauvin)

Fix check_hot_standby_delay perfdata output (Nicolas Thauvin)

Look in the correct place for the .ready files with the archive_ready action (Nicolas T

New action: commitratio (Guillaume Lelarge)

New action: hitratio (Guillaume Lelarge)

Make sure -action overrides the symlink naming trick.

Set defaults for archive_ready and wal_files (Thomas Guettler, GSM)

Better output for wal_files and archive_ready (GSM)

Fix warning when client_port set to empty string (bug #79)

Account for "empty row" in -x output (i.e. source of functions).

Fix some incorrectly named data fields (Andy Lester)

Expand the number of pgbouncer actions (Ruslan Kabalin)
```

```
Give detailed information and refactor txn_idle, txn_time, and query_time
    (Per request from bug #61)

Set maxalign to 8 in the bloat check if box identified as '64-bit'
    (Michel Sijmons, bug #66)

Support non-standard version strings in the bloat check.
    (Michel Sijmons and Gurjeet Singh, bug #66)

Do not show excluded databases in some output (Ruslan Kabalin)

Allow "and", "or" inside arguments (David E. Wheeler)

Add the "new_version_box" action.

Fix psql version regex (Peter Eisentraut, bug #69)

Add the -assume-standby-mode option (Ruslan Kabalin)

Note that txn_idle and query_time require 8.3 (Thomas Guettler)

Standardize and clean up all perfdata output (bug #52)

Exclude "idle in transaction" from the query_time check (bug #43)

Fix the perflimit for the bloat action (bug #50)

Clean up the custom_query action a bit.

Fix space in perfdata for hot_standby_delay action (Nicolas Thauvin)

Handle undef percents in check_fsm_relations (Andy Lester)

Fix typo in dbstats action (Stas Vitkovsky)

Fix MRTG for last vacuum and last_analyze actions.
```

Version 2.17.0 no public release

Version 2.16.0 January 20, 2011

```
  Add new action 'hot_standby_delay' (Nicolas Thauvin)
  Add cache-busting for the version-grabbing utilities.
  Fix problem with going to next method for new_version_pg
    (Greg Sabino Mullane, reported by Hywel Mallett in bug #65)
  Allow /usr/local/etc as an alternative location for the
```

```
  check_postgresrc file (Hywel Mallett)
Do not use tgisconstraint in same_schema if Postgres >= 9
  (Guillaume Lelarge)
```

Version 2.15.4 January 3, 2011

```
Fix warning when using symlinks
  (Greg Sabino Mullane, reported by Peter Eisentraut in bug #63)
```

Version 2.15.3 December 30, 2010

```
Show OK for no matching txn_idle entries.
```

Version 2.15.2 December 28, 2010

```
Better formatting of sizes in the bloat action output.

Remove duplicate perfs in bloat action output.
```

Version 2.15.1 December 27, 2010

```
Fix problem when examining items in pg_settings (Greg Sabino Mullane)

For connection test, return critical, not unknown, on FATAL errors
  (Greg Sabino Mullane, reported by Peter Eisentraut in bug #62)
```

Version 2.15.0 November 8, 2010

```
Add -quiet argument to suppress output on OK Nagios results
Add index comparison for same_schema (Norman Yamada and Greg Sabino Mullane)
Use $ENV{PGSERVICE} instead of "service=" to prevent problems (Guillaume Lelarge)
Add -man option to show the entire manual. (Andy Lester)
Redo the internal run_command() sub to use -x and hashes instead of regexes.
Fix error in custom logic (Andreas Mager)
Add the "pgbouncer_checksum" action (Guillaume Lelarge)
Fix regex to work on WIN32 for check_fsm_relations and check_fsm_pages (Luke Koops)
Don't apply a LIMIT when using -exclude on the bloat action (Marti Raudsepp)
Change the output of query_time to show pid,user,port, and address (Giles Westwood)
Fix to show database properly when using slony_status (Guillaume Lelarge)
Allow warning items for same_schema to be comma-separated (Guillaume Lelarge)
Constraint definitions across Postgres versions match better in same_schema.
Work against "EnterpriseDB" databases (Sivakumar Krishnamurthy and Greg Sabino Mullan
Separate perfdata with spaces (Jehan-Guillaume (ioguix) de Rorthais)
Add new action "archive_ready" (Jehan-Guillaume (ioguix) de Rorthais)
```

Version 2.14.3 (March 1, 2010)

```
Allow slony_status action to handle more than one slave.
Use commas to separate function args in same_schema output (Robert Treat)
```

Version 2.14.2 (February 18, 2010)

```
Change autovac_freeze default warn/critical back to 90%/95% (Robert Treat)
Put all items one-per-line for relation size actions if -verbose=1
```

Version 2.14.1 (February 17, 2010)

```
Don't use $^T in logfile check, as script may be long-running
Change the error string for the logfile action for easier exclusion
  by programs like tail_n_mail
```

Version 2.14.0 (February 11, 2010)

```
Added the 'slony_status' action.
Changed the logfile sleep from 0.5 to 1, as 0.5 gets rounded to 0 on some boxes!
```

Version 2.13.2 (February 4, 2010)

```
Allow timeout option to be used for logtime 'sleep' time.
```

Version 2.13.2 (February 4, 2010)

```
Show offending database for query_time action.
Apply perflimit to main output for sequence action.
Add 'noowner' option to same_schema action.
Raise sleep timeout for logfile check to 15 seconds.
```

Version 2.13.1 (February 2, 2010)

```
Fix bug preventing column constraint differences from 2 > 1 for same_schema from bein
Allow aliases 'dbname1', 'dbhost1', 'dbport1',etc.
Added "nolanguage" as a filter for the same_schema option.
Don't track "generic" table constraints (e.. $1, $2) using same_schema
```

Version 2.13.0 (January 29, 2010)

```
Allow "nofunctions" as a filter for the same_schema option.
Added "noperm" as a filter for the same_schema option.
Ignore dropped columns when considered positions for same_schema (Guillaume Lelarge)
```

Version 2.12.1 (December 3, 2009)

```
Change autovac_freeze default warn/critical from 90%/95% to 105%/120% (Marti Raudsepp
```

Version 2.12.0 (December 3, 2009)

```
Allow the temporary directory to be specified via the "tempdir" argument,
  for systems that need it (e.g. /tmp is not owned by root).
Fix so old versions of Postgres (< 8.0) use the correct default database (Giles Westw
For "same_schema" trigger mismatches, show the attached table.
Add the new_version_bc check for Bucardo version checking.
Add database name to perf output for last_vacuum|analyze (Guillaume Lelarge)
Fix for bloat action against old versions of Postgres without the 'block_size' param.
```

Version 2.11.1 (August 27, 2009)

Proper Nagios output for last_vacuum|analyze actions. (Cédric Villemain)
Proper Nagios output for locks action. (Cédric Villemain)
Proper Nagios output for txn_wraparound action. (Cédric Villemain)
Fix for constraints with embedded newlines for same_schema.
Allow -exclude for all items when using same_schema.

Version 2.11.0 (August 23, 2009)

Add Nagios perf output to the wal_files check (Cédric Villemain)
Add support for .check_postgresrc, per request from Albe Laurenz.
Allow list of web fetch methods to be changed with the -get_method option.
Add support for the -language argument, which overrides any ENV.
Add the -no-check_postgresrc flag.
Ensure check_postgresrc options are completely overridden by command-line options.
Fix incorrect warning > critical logic in replicate_rows (Glyn Astill)

Version 2.10.0 (August 3, 2009)

For same_schema, compare view definitions, and compare languages.
Make script into a global executable via the Makefile.PL file.
Better output when comparing two databases.
Proper Nagios output syntax for autovac_freeze and backends checks (Cédric Villemain)

Version 2.9.5 (July 24, 2009)

Don't use a LIMIT in check_bloat if -include is used. Per complaint from Jeff Frost.

Version 2.9.4 (July 21, 2009)

More French translations (Guillaume Lelarge)

Version 2.9.3 (July 14, 2009)

Quote dbname in perf output for the backends check. (Davide Abrigo)
Add 'fetch' as an alternative method for new_version checks, as this
  comes by default with FreeBSD. (Hywel Mallett)

Version 2.9.2 (July 12, 2009)

Allow dots and dashes in database name for the backends check (Davide Abrigo)
Check and display the database for each match in the bloat check (Cédric Villemain)
Handle 'too many connections' FATAL error in the backends check with a critical,
  rather than a generic error (Greg, idea by Jürgen Schulz-Brüssel)
Do not allow perflimit to interfere with exclusion rules in the vacuum and
  analyze tests. (Greg, bug reported by Jeff Frost)

Version 2.9.1 (June 12, 2009)

Fix for multiple databases with the check_bloat action (Mark Kirkwood)
Fixes and improvements to the same_schema action (Jeff Boes)
Write tests for same_schema, other minor test fixes (Jeff Boes)

Version 2.9.0 (May 28, 2009)

Added the same_schema action (Greg)

Version 2.8.1 (May 15, 2009)

Added timeout via statement_timeout in addition to perl alarm (Greg)

Version 2.8.0 (May 4, 2009)

Added internationalization support (Greg)
Added the 'disabled_triggers' check (Greg)
Added the 'prepared_txns' check (Greg)
Added the 'new_version_cp' and 'new_version_pg' checks (Greg)
French translations (Guillaume Lelarge)
Make the backends search return ok if no matches due to inclusion rules,
  per report by Guillaume Lelarge (Greg)
Added comprehensive unit tests (Greg, Jeff Boes, Selena Deckelmann)
Make fsm_pages and fsm_relations handle 8.4 servers smoothly. (Greg)
Fix missing 'upd' field in show_dbstats (Andras Fabian)
Allow ENV{PGCONTROLDATA} and ENV{PGBINDIR}. (Greg)
Add various Perl module infrastructure (e.g. Makefile.PL) (Greg)
Fix incorrect regex in txn_wraparound (Greg)
For txnwraparound: consistent ordering and fix duplicates in perf output (Andras Fabi
Add in missing exabyte regex check (Selena Deckelmann)
Set stats to zero if we bail early due to USERWHERECLAUSE (Andras Fabian)
Add additional items to dbstats output (Andras Fabian)
Remove -schema option from the fsm checks. (Greg Mullane and Robert Treat)
Handle case when ENV{PGUSER} is set. (Andy Lester)
Many various fixes. (Jeff Boes)
Fix -dbservice: check version and use ENV{PGSERVICE} for old versions (Cédric Villema

Version 2.7.3 (February 10, 2009)

Make the sequence action check if sequence being used for a int4 column and
react appropriately. (Michael Glaesemann)

Version 2.7.2 (February 9, 2009)

Fix to prevent multiple groupings if db arguments given.

Version 2.7.1 (February 6, 2009)

Allow the -p argument for port to work again.

Version 2.7.0 (February 4, 2009)

Do not require a connection argument, but use defaults and ENV variables when
  possible: PGHOST, PGPORT, PGUSER, PGDATABASE.

Version 2.6.1 (February 4, 2009)

Only require Date::Parse to be loaded if using the checkpoint action.

Version 2.6.0 (January 26, 2009)

Add the ‘checkpoint’ action.

Version 2.5.4 (January 7, 2009)

Better checking of $opt{dbservice} structure (Cédric Villemain)
Fix time display in timesync action output (Selena Deckelmann)
Fix documentation typos (Josh Tolley)

Version 2.5.3 (December 17, 2008)

Minor fix to regex in verify_version (Lee Jensen)

Version 2.5.2 (December 16, 2008)

Minor documentation tweak.

Version 2.5.1 (December 11, 2008)

Add support for -noidle flag to prevent backends action from counting idle processes.
Patch by Selena Deckelmann.

Fix small undefined warning when not using -dbservice.

Version 2.5.0 (December 4, 2008)

Add support for the pg_Service.conf file with the -dbservice option.

Version 2.4.3 (November 7, 2008)

Fix options for replicate_row action, per report from Jason Gordon.

Version 2.4.2 (November 6, 2008)

Wrap File::Temp::cleanup() calls in eval, in case File::Temp is an older version.
Patch by Chris Butler.

Version 2.4.1 (November 5, 2008)

Cast numbers to numeric to support sequences ranges > bigint in check_sequence action
Thanks to Scott Marlowe for reporting this.

Version 2.4.0 (October 26, 2008)

Add Cacti support with the dbstats action.
Pretty up the time output for last vacuum and analyze actions.
Show the percentage of backends on the check_backends action.

Version 2.3.10 (October 23, 2008)

Fix minor warning in action check_bloat with multiple databases.
Allow warning to be greater than critical when using the -reverse option.
Support the -perflimit option for the check_sequence action.

Version 2.3.9 (October 23, 2008)

Minor tweak to way we store the default port.

Version 2.3.8 (October 21, 2008)

Allow the default port to be changed easily.
Allow transform of simple output by MB, GB, etc.

Version 2.3.7 (October 14, 2008)

Allow multiple databases in 'sequence' action. Reported by Christoph Zwerschke.

Version 2.3.6 (October 13, 2008)

Add missing $schema to check_fsm_pages. (Robert Treat)

Version 2.3.5 (October 9, 2008)

Change option 'checktype' to 'valtype' to prevent collisions with -c[ritical]
Better handling of errors.

Version 2.3.4 (October 9, 2008)

Do explicit cleanups of the temp directory, per problems reported by sb@nnx.com.

Version 2.3.3 (October 8, 2008)

Account for cases where some rounding queries give -0 instead of 0.
Thanks to Glyn Astill for helping to track this down.

Version 2.3.2 (October 8, 2008)

Always quote identifiers in check_replicate_row action.

Version 2.3.1 (October 7, 2008)

Give a better error if one of the databases cannot be reached.

Version 2.3.0 (October 4, 2008)

Add the "sequence" action, thanks to Gavin M. Roy for the idea.
Fix minor problem with autovac_freeze action when using MRTG output.
Allow output argument to be case-insensitive.
Documentation fixes.

Version 2.2.4 (October 3, 2008)

Fix some minor typos

Version 2.2.3 (October 1, 2008)

Expand range of allowed names for -repinfo argument (Glyn Astill)
Documentation tweaks.

Version 2.2.2 (September 30, 2008)

Fixes for minor output and scoping problems.

Version 2.2.1 (September 28, 2008)

Add MRTG output to fsm_pages and fsm_relations.
Force error messages to one-line for proper Nagios output.
Check for invalid prereqs on failed command. From conversations with Euler Taveira de
Tweak the fsm_pages formula a little.

Version 2.2.0 (September 25, 2008)

Add fsm_pages and fsm_relations actions. (Robert Treat)

Version 2.1.4 (September 22, 2008)

Fix for race condition in txn_time action.
Add -debugoutput option.

Version 2.1.3 (September 22, 2008)

Allow alternate arguments "dbhost" for "host" and "dbport" for "port".
Output a zero as default value for second line of MRTG output.

Version 2.1.2 (July 28, 2008)

Fix sorting error in the "disk_space" action for non-Nagios output.
Allow -simple as a shortcut for -output=simple.

Version 2.1.1 (July 22, 2008)

Don't check databases with datallowconn false for the "autovac_freeze" action.

Version 2.1.0 (July 18, 2008)

Add the "autovac_freeze" action, thanks to Robert Treat for the idea and design.
Put an ORDER BY on the "txn_wraparound" action.

Version 2.0.1 (July 16, 2008)

Optimizations to speed up the "bloat" action quite a bit.
Fix "version" action to not always output in mrtg mode.

Version 2.0.0 (July 15, 2008)

Add support for MRTG and "simple" output options.
Many small improvements to nearly all actions.

Version 1.9.1 (June 24, 2008)

Fix an error in the bloat SQL in 1.9.0
Allow percentage arguments to be over 99%
Allow percentages in the bloat -warning and -critical (thanks to Robert Treat for the

Version 1.9.0 (June 22, 2008)

Don't include information_schema in certain checks. (Jeff Frost)
Allow -include and -exclude to use schemas by using a trailing period.

Version 1.8.5 (June 22, 2008)

Output schema name before table name where appropriate.
Thanks to Jeff Frost.

Version 1.8.4 (June 19, 2008)

Better detection of problems in -replicate_row.

Version 1.8.3 (June 18, 2008)

Fix 'backends' action: there may be no rows in pg_stat_activity, so run a second
query if needed to find the max_connections setting.
Thanks to Jeff Frost for the bug report.

Version 1.8.2 (June 10, 2008)

Changes to allow working under Nagios' embedded Perl mode. (Ioannis Tambouras)

Version 1.8.1 (June 9, 2008)

Allow 'bloat' action to work on Postgres version 8.0.
Allow for different commands to be run for each action depending on the server version
Give better warnings when running actions not available on older Postgres servers.

Version 1.8.0 (June 3, 2008)

Add the -reverse option to the custom_query action.

Version 1.7.1 (June 2, 2008)

Fix 'query_time' action: account for race condition in which zero rows appear in pg_st
Thanks to Dustin Black for the bug report.

Version 1.7.0 (May 11, 2008)

Add -replicate_row action

Version 1.6.1 (May 11, 2008)

Add -symlinks option as a shortcut to -action=rebuild_symlinks

Version 1.6.0 (May 11, 2008)

Add the custom_query action.

Version 1.5.2 (May 2, 2008)

Fix problem with too eager creation of custom pgpass file.

Version 1.5.1 (April 17, 2008)

Add example Nagios configuration settings (Brian A. Seklecki)

Version 1.5.0 (April 16, 2008)

Add the -includeuser and -excludeuser options. Documentation cleanup.

Version 1.4.3 (April 16, 2008)

Add in the 'output' concept for future support of non-Nagios programs.

Version 1.4.2 (April 8, 2008)

```
Fix bug preventing -dbpass argument from working (Robert Treat).
```

Version 1.4.1 (April 4, 2008)

```
Minor documentation fixes.
```

Version 1.4.0 (April 2, 2008)

```
Have 'wal_files' action use pg_ls_dir (idea by Robert Treat).
For last_vacuum and last_analyze, respect autovacuum effects, add separate
  autovacuum checks (ideas by Robert Treat).
```

Version 1.3.1 (April 2, 2008)

```
Have txn_idle use query_start, not xact_start.
```

Version 1.3.0 (March 23, 2008)

```
Add in txn_idle and txn_time actions.
```

Version 1.2.0 (February 21, 2008)

```
Add the 'wal_files' action, which counts the number of WAL files
  in your pg_xlog directory.
Fix some typos in the docs.
Explicitly allow -v as an argument.
Allow for a null syslog_facility in the 'logfile' action.
```

Version 1.1.2 (February 5, 2008)

```
Fix error preventing -action=rebuild_symlinks from working.
```

Version 1.1.1 (February 3, 2008)

```
Switch vacuum and analyze date output to use 'DD', not 'D'. (Glyn Astill)
```

Version 1.1.0 (December 16, 2008)

```
Fixes, enhancements, and performance tracking.
Add performance data tracking via -showperf and -perflimit
Lots of refactoring and cleanup of how actions handle arguments.
Do basic checks to figure out syslog file for 'logfile' action.
Allow for exact matching of beta versions with 'version' action.
Redo the default arguments to only populate when neither 'warning' nor 'critical' is p
Allow just warning OR critical to be given for the 'timesync' action.
Remove 'redirect_stderr' requirement from 'logfile' due to 8.3 changes.
Actions 'last_vacuum' and 'last_analyze' are 8.2 only (Robert Treat)
```

Version 1.0.16 (December 7, 2007)

```
First public release, December 2007
```

BUGS AND LIMITATIONS

The index bloat size optimization is rough.

Some actions may not work on older versions of Postgres (before 8.0).

Please report any problems to check_postgres@bucardo.org

AUTHOR

Greg Sabino Mullane greg@turnstep.com

NAGIOS EXAMPLES

Some example Nagios configuration settings using this script:

```
 define command {
     command_name    check_postgres_size
     command_line    $USER2$/check_postgres.pl -H $HOSTADDRESS$ -u pgsql -db postgres -actio
 }

define command {
     command_name    check_postgres_locks
     command_line    $USER2$/check_postgres.pl -H $HOSTADDRESS$ -u pgsql -db postgres -actio
 }

define service {
     use                     generic-other
     host_name               dbhost.gtld
     service_description     dbhost PostgreSQL Service Database Usage Size
     check_command           check_postgres_size!256000000!512000000
 }

define service {
     use                     generic-other
     host_name               dbhost.gtld
     service_description     dbhost PostgreSQL Service Database Locks
     check_command           check_postgres_locks!2!3
 }
```

LICENSE AND COPYRIGHT

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.