pgBadger - A fast PostgreSQL Log Analyzer

NAME

pgBadger - a fast PostgreSQL log analysis report

SYNOPSIS

Usage: pgbadger [options] logfile [...]

PostgreSQL log analyzer with fully detailed reports and graphs.

Arguments:

logfile can be a single log file, a list of files, or a shell command
returning a list of files. If you want to pass log content from stdin
use - as filename. Note that input from stdin will not work with csvlog.

Options:

```
-a | --average minutes : number of minutes to build the average graphs of
                         queries and connections. Default 5 minutes.
-A | --histo-average min: number of minutes to build the histogram graphs
                         of queries. Default 60 minutes.
-b | --begin datetime  : start date/time for the data to be parsed in log
                         (either a timestamp or a time)
-c | --dbclient host   : only report on entries for the given client host.
-C | --nocomment       : remove comments like /* ... */ from queries.
-d | --dbname database : only report on entries for the given database.
-D | --dns-resolv      : client ip addresses are replaced by their DNS name.
                         Be warned that this can really slow down pgBadger.
-e | --end datetime    : end date/time for the data to be parsed in log
                         (either a timestamp or a time)
-E | --explode         : explode the main report by generating one report
                         per database. Global information not related to a
                         database are added to the postgres database report.
-f | --format logtype  : possible values: syslog, syslog2, stderr, jsonlog,
                         cvs, pgbouncer, logplex, rds and redshift. Use this
                         option when pgBadger is not able to detect the log
                         format.
-G | --nograph         : disable graphs on HTML output. Enabled by default.
-h | --help            : show this message and exit.
-H | --html-dir path   : path to directory where HTML report must be written
                         in incremental mode, binary files stay on directory
                         defined with -O, --outdir option.
-i | --ident name      : programname used as syslog ident. Default: postgres
-I | --incremental     : use incremental mode, reports will be generated by
                         days in a separate directory, --outdir must be set.
-j | --jobs number     : number of jobs to run at same time. Run as single
                         by default or when working with csvlog.
-J | --Jobs number     : number of log file to parse in parallel. Process one
                         file at a time by default or when csvlog is used.
-l | --last-parsed file: allow incremental log parsing by registering the
                         last datetime and line parsed. Useful if you want
                         to watch errors since last run or if you want one
                         report per day with a log rotated each week.
-L | --logfile-list file:file containing a list of log file to parse.
-m | --maxlength size  : maximum length of a query, it will be restricted to
                         the given size. Default truncate size is 100000.
-M | --no-multiline    : do not collect multiline statement to avoid garbage
                         especially on errors that generate a huge report.
-n | --nohighlight     : disable SQL code highlighting.
-N | --appname name    : only report on entries for given application name
-o | --outfile filename: define the filename for the output. Default depends
                         on the output format: out.html, out.txt, out.bin,
                         out.json or out.tsung. This option can be used
                         multiple time to output several format. To use json
                         output the Perl module JSON::XS must be installed,
                         To dump output to stdout use - as filename.
-O | --outdir path     : directory where out file must be saved.
-p | --prefix string   : the value of your custom log_line_prefix as
                         defined in your postgresql.conf. Only use it if you
                         aren't using one of the standard prefixes specified
                         in the pgBadger documentation, such as if your
                         prefix includes additional variables like client ip
                         or application name. See examples below.
-P | --no-prettify     : disable SQL queries prettify formatter.
-q | --quiet           : don't print anything to stdout, not even a progress
                         bar.
-r | --remote-host ip  : set the host where to execute the cat command on
                         remote logfile to parse locally the file.
-R | --retention N     : number of weeks to keep in incremental mode. Default
                         to 0, disabled. Used to set the number of weeks to
                         keep in output directory. Older weeks and days
                         directory are automatically removed.
-s | --sample number   : number of query samples to store. Default: 3.
-S | --select-only     : only report SELECT queries.
-t | --top number      : number of queries to store/display. Default: 20.
-T | --title string    : change title of the HTML page report.
-u | --dbuser username : only report on entries for the given user.
-U | --exclude-user username : exclude entries for the specified user from
                         report. Can be used multiple time.
-v | --verbose         : enable verbose or debug mode. Disabled by default.
```

```
  -V | --version         : show pgBadger version and exit.
  -w | --watch-mode      : only report errors just like logwatch could do.
  -W | --wide-char       : encode html output of queries into UTF8 to avoid
                           Perl message "Wide character in print".
  -x | --extension       : output format. Values: text, html, bin, json or
                           tsung. Default: html
  -X | --extra-files     : in incremental mode allow pgBadger to write CSS and
                           JS files in the output directory as separate files.
  -z | --zcat exec_path  : set the full path to the zcat program. Use it if
                           zcat or bzcat or unzip is not in your path.
  -Z | --timezone +/-XX  : Set the number of hours from GMT of the timezone.
                           Use this to adjust date/time in JavaScript graphs.
  --pie-limit num        : pie data lower than num% will show a sum instead.
  --exclude-query regex  : any query matching the given regex will be excluded
                           from the report. For example: "^(VACUUM|COMMIT)"
                           You can use this option multiple times.
  --exclude-file filename: path of the file which contains all the regex to
                           use to exclude queries from the report. One regex
                           per line.
  --include-query regex  : any query that does not match the given regex will
                           be excluded from the report. You can use this
                           option multiple times. For example: "(tbl1|tbl2)".
  --include-file filename: path of the file which contains all the regex of
                           the queries to include from the report. One regex
                           per line.
  --disable-error        : do not generate error report.
  --disable-hourly       : do not generate hourly report.
  --disable-type         : do not generate report of queries by type, database
                           or user.
  --disable-query        : do not generate query reports (slowest, most
                           frequent, queries by users, by database, ...).
  --disable-session      : do not generate session report.
  --disable-connection   : do not generate connection report.
  --disable-lock         : do not generate lock report.
  --disable-temporary    : do not generate temporary report.
  --disable-checkpoint   : do not generate checkpoint/restartpoint report.
  --disable-autovacuum   : do not generate autovacuum report.
  --charset              : used to set the HTML charset to be used.
                           Default: utf-8.
  --csv-separator        : used to set the CSV field separator, default: ,
  --exclude-time  regex  : any timestamp matching the given regex will be
                           excluded from the report. Example: "2013-04-12 .*"
                           You can use this option multiple times.
  --include-time  regex  : only timestamps matching the given regex will be
                           included in the report. Example: "2013-04-12 .*"
                           You can use this option multiple times.
  --exclude-db name      : exclude entries for the specified database from
                           report. Example: "pg_dump". Can be used multiple
                           time.
  --exclude-appname name : exclude entries for the specified application name
                           from report.  Example: "pg_dump".  Can be used
                           multiple time.
  --exclude-line regex   : pgBadger will start to exclude any log entry that
                           will match the given regex. Can be used multiple
                           time.
  --exclude-client name  : exclude log entries for the specified client ip.
                           Can be used multiple time.
  --anonymize            : obscure all literals in queries, useful to hide
                           confidential data.
  --noreport             : prevent pgBadger to create reports in incremental
                           mode.
  --log-duration         : force pgBadger to associate log entries generated
                           by both log_duration = on and log_statement = 'all'
  --enable-checksum      : used to add a md5 sum under each query report.
  --journalctl command   : command to use to replace PostgreSQL logfile by
                           a call to journalctl. Basically it might be:
                             journalctl -u postgresql-9.5
  --pid-dir path         : set the path where the pid file must be stored.
                           Default /tmp
  --pid-file file        : set the name of the pid file to manage concurrent
                           execution of pgBadger. Default: pgbadger.pid
  --rebuild              : used to rebuild all html reports in incremental
                           output directories where there's binary data files.
  --pgbouncer-only       : only show PgBouncer related menu in the header.
  --start-monday         : in incremental mode, calendar's weeks start on
                           sunday. Use this option to start on monday.
  --normalized-only      : only dump all normalized query to out.txt
  --log-timezone +/-XX   : Set the number of hours from GMT of the timezone
                           that must be used to adjust date/time read from
                           log file before beeing parsed. Using this option
                           make more difficult log search with a date/time.
  --prettify-json        : use it if you want json output to be prettified.
  --month-report YYYY-MM : create a cumulative HTML report over the specified
                           month. Requires incremental output directories and
                           the presence of all necessary binary data files
  --noexplain            : do not process lines generated by auto_explain.
  --command CMD          : command to execute to retrieve log entries on
                           stdin. pgBadger will open a pipe to the command
                           and parse log entries generated by the command.
  --no-week              : inform pgbadger to not build weekly reports in
                           incremental mode. Useful if it takes too much time.
  --explain-url URL      : use it to override the url of the graphical explain
                           tool. Default: http://explain.depesz.com/?is_public=0&is_anon=0&plan=
```

pgBadger is able to parse a remote log file using a passwordless ssh connection. Use the -r or --remote-host to set the host ip address or hostname. There's also some additional options to fully control the ssh connection.

```
--ssh-program ssh   path to the ssh program to use. Default: ssh.
--ssh-user username      connection login name. Default to running user.
--ssh-identity file      path to the identity file to use.
--ssh-timeout second     timeout to ssh connection failure. Default 10 secs.
--ssh-option  options    list of -o options to use for the ssh connection.
                      Options always used:
                         -o ConnectTimeout=$ssh_timeout
                         -o PreferredAuthentications=hostbased,publickey
```

Log file to parse can also be specified using an URI, supported protocol are http[s] and [s]ftp. The curl command will be used to download the file and the file will be parsed during download. The ssh protocol is also supported and will use the ssh command like with the remote host use. See examples bellow.

Examples:

```
pgbadger /var/log/postgresql.log
pgbadger /var/log/postgres.log.2.gz /var/log/postgres.log.1.gz /var/log/postgres.log
pgbadger /var/log/postgresql/postgresql-2012-05-*
pgbadger --exclude-query="^(COPY|COMMIT)" /var/log/postgresql.log
pgbadger -b "2012-06-25 10:56:11" -e "2012-06-25 10:59:11" /var/log/postgresql.log
cat /var/log/postgres.log | pgbadger -
# Log prefix with stderr log output
pgbadger --prefix '%t [%p]: user=%u,db=%d,client=%h' /pglog/postgresql-2012-08-21*
pgbadger --prefix '%m %u@%d %p %r %a : ' /pglog/postgresql.log
# Log line prefix with syslog log output
pgbadger --prefix 'user=%u,db=%d,client=%h,appname=%a' /pglog/postgresql-2012-08-21*
# Use my 8 CPUs to parse my 10GB file faster, much faster
pgbadger -j 8 /pglog/postgresql-10.1-main.log
```

Use URI notation for remote log file:

```
pgbadger http://172.12.110.1//var/log/postgresql/postgresql-10.1-main.log
pgbadger ftp://username@172.12.110.14/postgresql-10.1-main.log
pgbadger ssh://username@172.12.110.14//var/log/postgresql/postgresql-10.1-main.log*
```

You can use together a local PostgreSQL log and a remote pgbouncer log file to parse:

```
pgbadger /var/log/postgresql/postgresql-10.1-main.log ssh://username@172.12.110.14/pgbouncer.log
```

Generate Tsung sessions XML file with select queries only:

```
pgbadger -S -o sessions.tsung --prefix '%t [%p]: user=%u,db=%d ' /pglog/postgresql-10.1.log
```

Reporting errors every week by cron job:

```
30 23 * * 1 /usr/bin/pgbadger -q -w /var/log/postgresql.log -o /var/reports/pg_errors.html
```

Generate report every week using incremental behavior:

```
0 4 * * 1 /usr/bin/pgbadger -q `find /var/log/ -mtime -7 -name "postgresql.log*"` -o /var/reports/pg_errors-`date +\%F`.html -l /var/reports/pgbadger_incremental_file.dat
```

This supposes that your log file and HTML report are also rotated every week.

Or better, use the auto-generated incremental reports:

```
0 4 * * * /usr/bin/pgbadger -l -q /var/log/postgresql/postgresql.log.1 -O /var/www/pg_reports/
```

will generate a report per day and per week.

In incremental mode, you can also specify the number of week to keep in the reports:

```
/usr/bin/pgbadger --retention 2 -l -q /var/log/postgresql/postgresql.log.1 -O /var/www/pg_reports/
```

If you have a pg_dump at 23:00 and 13:00 each day during half an hour, you can use pgBadger as follow to exclude these period from the report:

```
pgbadger --exclude-time "2013-09-.* (23|13):.*" postgresql.log
```

This will help avoid having COPY statements, as generated by pg_dump, on top of the list of slowest queries. You can also use --exclude-appname "pg_dump" to solve this problem in a simpler way.

You can also parse journalctl output just as if it was a log file:

```
pgbadger --journalctl 'journalctl -u postgresql-9.5'
```

or worst, call it from a remote host:

```
pgbadger -r 192.168.1.159 --journalctl 'journalctl -u postgresql-9.5'
```

you don't need to specify any log file at command line, but if you have other PostgreSQL log file to parse, you can add them as usual.

To rebuild all incremental html reports after, proceed as follow:

```
rm /path/to/reports/*.js
rm /path/to/reports/*.css
pgbadger -X -l -O /path/to/reports/ --rebuild
```

it will also update all resource files (JS and CSS). Use -E or --explode if the reports were built using this option.

pgBadger also support Heroku PostgreSQL logs using logplex format:

```
heroku logs -p postgres | pgbadger -f logplex -o heroku.html -
```

this will stream Heroku PostgreSQL log to pgbadger through stdin.

pgBadger can auto detect RDS and cloudwatch PostgreSQL logs using rds format:

```
pgbadger -f rds -o rds_out.html rds.log
```

To create a cumulative report over a month use command:

```
pgbadger --month-report 2919-05 /path/to/incremantal/reports/
```

this will add a link to the month name into the calendar view in incremental reports to look at report for month 2019 May. Use -E or --explode if the reports were built using this option.

DESCRIPTION

pgBadger is a PostgreSQL log analyzer built for speed with fully reports from your PostgreSQL log file. It's a single and small Perl script that outperforms any other PostgreSQL log analyzer.

It is written in pure Perl and uses a JavaScript library (flotr2) to draw graphs so that you don't need to install any additional Perl modules or other packages. Furthermore, this library gives us more features such as zooming. pgBadger also uses the Bootstrap JavaScript library and the FontAwesome webfont for better design. Everything is embedded.

pgBadger is able to autodetect your log file format (syslog, stderr, csvlog or jsonlog) if the file is long enough. It is designed to parse huge log files as well as compressed files. Supported compressed format are gzip, bzip2, lz4, xz, zip and zstd. For the xz format you must have an xz version upper than 5.05 that supports the --robot option. In order pgbadger determine uncompressed file size with lz4, file must be compressed with --content-size option. For the complete list of features see below.

All charts are zoomable and can be saved as PNG images.

You can also limit pgBadger to only report errors or remove any part of the report using command line options.

pgBadger supports any custom format set into the log_line_prefix directive of your postgresql.conf file as long as it at least specify the %t and %p patterns.

pgBadger allows parallel processing of a single log file or multiple files through the use of the -j option specifying the number of CPUs.

If you want to save system performance you can also use log_duration instead of log_min_duration_statement to have reports on duration and number of queries only.

FEATURE

pgBadger reports everything about your SQL queries:

- Overall statistics.
- The most frequent waiting queries.
- Queries that waited the most.
- Queries generating the most temporary files.
- Queries generating the largest temporary files.
- The slowest queries.
- Queries that took up the most time.
- The most frequent queries.
- The most frequent errors.
- Histogram of query times.
- Histogram of sessions times.
- Users involved in top queries.
- Applications involved in top queries.
- Queries generating the most cancellation.
- Queries most cancelled.
- The most time consuming prepare/bind queries

The following reports are also available with hourly charts divided into periods of five minutes:

- SQL queries statistics.
- Temporary file statistics.
- Checkpoints statistics.
- Autovacuum and autoanalyze statistics.
- Cancelled queries.
- Error events (panic, fatal, error and warning).
- Error class distribution.

There are also some pie charts about distribution of:

- Locks statistics.
- Queries by type (select/insert/update/delete).
- Distribution of queries type per database/application
- Sessions per database/user/client/application.
- Connections per database/user/client/application.
- Autovacuum and autoanalyze per table.
- Queries per user and total duration per user.

All charts are zoomable and can be saved as PNG images. SQL queries reported are highlighted and beautified automatically.

pgBadger is also able to parse PgBouncer log files and to create the following reports:

- Request Throughput
- Bytes I/O Throughput
- Queries Average duration
- Simultaneous sessions
- Histogram of sessions times
- Sessions per database
- Sessions per user
- Sessions per host
- Established connections
- Connections per database

Connections per user
Connections per host
Most used reserved pools
Most Frequent Errors/Events

You can also have incremental reports with one report per day and a cumulative report per week. Two multiprocess modes are available to speed up log parsing, one using one core per log file, and the second using multiple cores to parse a single file. These modes can be combined.

Histogram granularity can be adjusted using the -A command line option. By default they will report the mean of each top queries/errors occurring per hour, but you can specify the granularity down to the minute.

pgBadger can also be used in a central place to parse remote log files using a passwordless SSH connection. This mode can be used with compressed files and in the multiprocess per file mode (-J) but can not be used with the CSV log format.

REQUIREMENT

pgBadger comes as a single Perl script - you do not need anything other than a modern Perl distribution. Charts are rendered using a JavaScript library so you don't need anything other than a web browser. Your browser will do all the work.

If you planned to parse PostgreSQL CSV log files you might need some Perl Modules:

Text::CSV_XS - to parse PostgreSQL CSV log files.

This module is optional, if you don't have PostgreSQL log in the CSV format you don't need to install it.

If you want to export statistics as JSON file you need an additional Perl module:

JSON::XS - JSON serialising/deserialising, done correctly and fast

This module is optional, if you don't select the json output format you don't need to install it. You can install it on a Debian like system using:

sudo apt-get install libjson-xs-perl

and in RPM like system using:

sudo yum install perl-JSON-XS

Compressed log file format is autodetected from the file extension. If pgBadger find a gz extension it will use the zcat utility, with a bz2 extension it will use bzcat, with lz4 it will use lz4cat, with zst it will use zstdcat and if the file extension is zip or xz then the unzip or xz utilities will be used.

If those utilities are not found in the PATH environment variable then use the --zcat command line option to change this path. For example:

--zcat="/usr/local/bin/gunzip -c" or --zcat="/usr/local/bin/bzip2 -dc"
--zcat="C:\tools\unzip -p"

By default pgBadger will use the zcat, bzcat, lz4cat, zstdcat and unzip utilities following the file extension. If you use the default autodetection compress format you can mixed gz, bz2, lz4, xz, zip or zstd files. Specifying a custom value to --zcat option will remove this feature of mixed compressed format.

Note that multiprocessing can not be used with compressed files or CSV files as well as under Windows platform.

INSTALLATION

Download the tarball from GitHub and unpack the archive as follow:

tar xzf pgbadger-11.x.tar.gz
cd pgbadger-11.x/
perl Makefile.PL
make && sudo make install

This will copy the Perl script pgbadger to /usr/local/bin/pgbadger by default and the man page into /usr/local/share/man/man1/pgbadger.1. Those are the default installation directories for 'site' install.

If you want to install all under /usr/ location, use INSTALLDIRS='perl' as an argument of Makefile.PL. The script will be installed into /usr/bin/pgbadger and the manpage into /usr/share/man/man1/pgbadger.1.

For example, to install everything just like Debian does, proceed as follows:

perl Makefile.PL INSTALLDIRS=vendor

By default INSTALLDIRS is set to site.

POSTGRESQL CONFIGURATION

You must enable and set some configuration directives in your postgresql.conf before starting.

You must first enable SQL query logging to have something to parse:

log_min_duration_statement = 0

Here every statement will be logged, on a busy server you may want to increase this value to only log queries with a longer duration. Note that if you have log_statement set to 'all' nothing will be logged through the log_min_duration_statement directive. See the next chapter for more information.

pgBadger supports any custom format set into the log_line_prefix directive of your postgresql.conf file as long as it at least specify a time escape sequence (%t, %m or %n) and the process related escape sequence (%p or %c).

For example, with 'stderr' log format, log_line_prefix must be at least:

```
log_line_prefix = '%t [%p]: '
```

Log line prefix could add user, database name, application name and client ip address as follows:

```
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
```

or for syslog log file format:

```
log_line_prefix = 'user=%u,db=%d,app=%a,client=%h '
```

Log line prefix for stderr output could also be:

```
log_line_prefix = '%t [%p]: db=%d,user=%u,app=%a,client=%h '
```

or for syslog output:

```
log_line_prefix = 'db=%d,user=%u,app=%a,client=%h '
```

You need to enable other parameters in postgresql.conf to get more information from your log files:

```
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
```

Do not enable log_statement as its log format will not be parsed by pgBadger.

Of course your log messages should be in English with or without locale support:

```
lc_messages='en_US.UTF-8'
lc_messages='C'
```

pgBadger parser do not support other locale like 'fr_FR.UTF-8' for example.

LOG STATEMENTS

Considerations about log_min_duration_statement, log_duration and log_statement configuration directives.

If you want the query statistics to include the actual query strings, you must set log_min_duration_statement to 0 or more milliseconds.

If you just want to report duration and number of queries and don't want all details about queries, set log_min_duration_statement to -1 to disable it and enable log_duration in your postgresql.conf file. If you want to add the most common request report you can either choose to set log_min_duration_statement to a higher value or choose to enable log_statement.

Enabling log_min_duration_statement will add reports about slowest queries and queries that took up the most time. Take care that if you have log_statement set to 'all' nothing will be logged with log_min_duration_statement.

Warning: Do not enable both log_min_duration_statement, log_duration and log_statement all together, this will result in wrong counter values. Note that this will also increase drastically the size of your log. log_min_duration_statement should always be preferred.

PARALLEL PROCESSING

To enable parallel processing you just have to use the -j N option where N is the number of cores you want to use.

pgBadger will then proceed as follow:

```
for each log file
    chunk size = int(file size / N)
    look at start/end offsets of these chunks
    fork N processes and seek to the start offset of each chunk
       each process will terminate when the parser reach the end offset
       of its chunk
       each process write stats into a binary temporary file
   wait for all children has terminated
All binary temporary files generated will then be read and loaded into
memory to build the html output.
```

With that method, at start/end of chunks pgBadger may truncate or omit a maximum of N queries per log file which is an insignificant gap if you have millions of queries in your log file. The chance that the query that you were looking for is lost is near 0, this is why I think this gap is livable. Most of the time the query is counted twice but truncated.

When you have many small log files and many CPUs it is speedier to dedicate one core to one log file at a time. To enable this behavior you have to use option -J N instead. With 200 log files of 10MB each the use of the -J option starts being really interesting with 8 Cores. Using this method you will be sure not to lose any queries in the reports.

He are a benchmark done on a server with 8 CPUs and a single file of 9.5GB.

```
 Option |  1 CPU  | 2 CPU | 4 CPU | 8 CPU
--------+---------+-------+-------+------
   -j   | 1h41m18 | 50m25 | 25m39 | 15m58
   -J   | 1h41m18 | 54m28 | 41m16 | 34m45
```

With 200 log files of 10MB each and a total of 2GB the results are slightly different:

```
 Option | 1 CPU | 2 CPU | 4 CPU | 8 CPU
--------+-------+-------+-------+------
   -j   | 20m15 |  9m56 |  5m20 | 4m20
```

```
-J   | 20m15 |  9m49 |  5m00 | 2m40
```

So it is recommended to use -j unless you have hundreds of small log files and can use at least 8 CPUs.

IMPORTANT: when you are using parallel parsing pgBadger will generate a lot of temporary files in the /tmp directory and will remove them at the end, so do not remove those files unless pgBadger is not running. They are all named with the following template tmp_pgbadgerXXXX.bin so they can be easily identified.

INCREMENTAL REPORTS

pgBadger includes an automatic incremental report mode using option -I or --incremental. When running in this mode, pgBadger will generate one report per day and a cumulative report per week. Output is first done in binary format into the mandatory output directory (see option -O or --outdir), then in HTML format for daily and weekly reports with a main index file.

The main index file will show a dropdown menu per week with a link to each week report and links to daily reports of each week.

For example, if you run pgBadger as follows based on a daily rotated file:

```
0 4 * * * /usr/bin/pgbadger -I -q /var/log/postgresql/postgresql.log.1 -O /var/www/pg_reports/
```

you will have all daily and weekly reports for the full running period.

In this mode pgBadger will create an automatic incremental file in the output directory, so you don't have to use the -l option unless you want to change the path of that file. This means that you can run pgBadger in this mode each day on a log file rotated each week, and it will not count the log entries twice.

To save disk space you may want to use the -X or --extra-files command line option to force pgBadger to write JavaScript and CSS to separate files in the output directory. The resources will then be loaded using script and link tags.

Rebuilding reports

Incremental reports can be rebuilt after a pgbadger report fix or a new feature to update all HTML reports. To rebuild all reports where a binary file is still present proceed as follow:

```
rm /path/to/reports/*.js
rm /path/to/reports/*.css
pgbadger -X -I -O /path/to/reports/ --rebuild
```

it will also update all resource files (JS and CSS). Use -E or --explode if the reports were built using this option.

Monthly reports

By default pgBadger in incremental mode only compute daily and weekly reports. If you want monthly cumulative reports you will have to use a separate command to specify the report to build. For example to build a report for August 2019:

```
pgbadger -X --month-report 2919-08 /var/www/pg_reports/
```

this will add a link to the month name into the calendar view of incremental reports to look at monthly report. The report for a current month can be run every day it is entirely rebuilt each time. The monthly report is not built by default because it could take lot of time following the amount of data.

If reports were built with the per database option (-E | --explode) it must be used too when calling pgbadger to build monthly report:

```
pgbadger -E -X --month-report 2919-08 /var/www/pg_reports/
```

This is the same when using the rebuild option (-R | --rebuild).

BINARY FORMAT

Using the binary format it is possible to create custom incremental and cumulative reports. For example, if you want to refresh a pgBadger report each hour from a daily PostgreSQL log file, you can proceed by running each hour the following commands:

```
pgbadger --last-parsed .pgbadger_last_state_file -o sunday/hourX.bin /var/log/pgsql/postgresql-Sun.log
```

to generate the incremental data files in binary format. And to generate the fresh HTML report from that binary file:

```
pgbadger sunday/*.bin
```

Or as another example, if you generate one log file per hour and you want reports to be rebuilt each time the log file is rotated, proceed as follows:

```
pgbadger -o day1/hour01.bin /var/log/pgsql/pglog/postgresql-2012-03-23_10.log
pgbadger -o day1/hour02.bin /var/log/pgsql/pglog/postgresql-2012-03-23_11.log
pgbadger -o day1/hour03.bin /var/log/pgsql/pglog/postgresql-2012-03-23_12.log
...
```

When you want to refresh the HTML report, for example each time after a new binary file is generated, just do the following:

```
pgbadger -o day1_report.html day1/*.bin
```

Adjust the commands to suit your particular needs.

JSON FORMAT

JSON format is good for sharing data with other languages, which makes it easy to integrate pgBadger result into other monitoring tools like Cacti or Graphite.

AUTHORS

pgBadger is an original work from Gilles Darold.

The pgBadger logo is an original creation of Damien Cazeils.

The pgBadger v4.x design comes from the "Art is code" company.

This web site is a work of Gilles Darold.

pgBadger is maintained by Gilles Darold and every one who wants to contribute.

Many people have contributed to pgBadger, they are all quoted in the Changelog file.

LICENSE