

# HACKING ON POSTGRES

AN OVERVIEW

James Coleman (PGConf.NYC 2022)

# About Me

- Architect for Data Engineering at Braintree Payments
- Not the most senior hacker in the room (unless poorly attended!), but...I've authored patches in Postgres versions 12-15, including:
  - Incremental sort
  - Multiple improvements to `ScalarArrayOpExpr ([NOT] IN, = ANY/ALL)` optimization and execution.

# Presentation Note

- Many slides include footnote references to a number of links and file paths.
- The slides are already available as a PDF download on the conference website.

# Other Talks in this Genre

## Further material that complements this talk

- Hacking on Postgres (Stephen Frost) [1]
- Intro to Postgres Planner Hacking (Melanie Plageman) [2]
- How to be a Happy Hacker (Andrew Dunstan) [3]
- Other resources listed on the Wiki [4]

1. <https://www.postgresql.eu/events/pgconfeu2018/sessions/session/2058/slides/96/hackingpg-present.pdf>

2. <https://www.pgcon.org/2019/schedule/events/1379.en.html>

3. <https://www.youtube.com/watch?v=yFDyM29tB6k>

4. [https://wiki.postgresql.org/wiki/So,\\_you\\_want\\_to\\_be\\_a\\_developer%3F#Hacking\\_PostgreSQL\\_Resources](https://wiki.postgresql.org/wiki/So,_you_want_to_be_a_developer%3F#Hacking_PostgreSQL_Resources)

**What about this talk?**

**“I would love to see more talks about mechanics of working on the PG codebase...e.g. how to set up a feedback cycle, navigate in vim, etc.”**

**- Braintree colleague**

**“That is an under-appreciated superpower of Ruby development here...bootstrap your env, run focused tests from vimux...rinse, repeat...It'd be really cool to develop a ‘standard’ way of working with Postgres in vim, vscode, or whatever.”**

**- Braintree colleague**

# Roadmap

## Focus on mechanics of hacking on Postgres

Three broad categories:

- Community process
- Codebase
- Development tooling

# Community Process

**Mailing list, patch submission and review, and CommitFests**

# Mailing lists

- Development happens publicly on the pgsql-hackers [1] mailing list.
  - Some might also start on the pgsql-bugs [2] mailing list.
- Discussions can span multiple years and 100s of messages (this is why e.g. GitHub Pull Requests wouldn't work).
  - Don't get discouraged if a discussion takes a long time!
  - Many patches and ideas are rejected early.

1.<https://www.postgresql.org/list/pgsql-hackers/>

2.<https://www.postgresql.org/list/pgsql-bugs/>

# Mailing lists

- Don't top-post (use inline reply/interleave posting style)
- Use plain text
- Reply-all to the proper point in the thread tree

# Mailing lists

- I recommend subscribing to the list now to ensure you get all of the messages (including parts of a discussion tree you weren't cc'd on), and it's easy to reply.
- Hint: setup a filter to send all the messages into a specific folder or label.

Matches: **list:(<pgsql-hackers.lists.postgresql.org> -{jtc331@gmail.com})**  
Do this: Skip Inbox, Apply label "PG Hackers", Never send it to Spam

Matches: **list:<pgsql-hackers.lists.postgresql.org> jtc331@gmail.com**  
Do this: Apply label "PG Hackers", Never send it to Spam, Mark it as important

Matches: **to:(pgsql-hackers@postgresql.org jtc331@gmail.com)**  
Do this: Apply label "PG Hackers", Never send it to Spam, Mark it as important

# Mailing lists

- If you really don't want a firehose of emails, it's now possible (while logged in) to Resend a single message (to which you can then reply).
- You can also subscribe without receiving email (this allows sent emails to bypass moderation).

## Re: Reducing the chunk header sizes on all memory context types

From: Tomas Vondra <tomas(dot)vondra(at)enterprisedb(dot)com>  
To: David Rowley <dgrowleym1(at)gmail(dot)com>  
Cc: Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>, Amit Kapila <amit(dot)kapila16(at)gmail(dot)com>, Andres Freund <andres(at)anarazel(dot)de>, Robert Haas <robertmhaas(at)gmail(dot)com>, Yura Sokolov <y(dot)sokolov(at)postgrespro(dot)ru>, PostgreSQL Developers <pgsql-hackers(at)lists(dot)postgresql(dot)org>  
Subject: Re: Reducing the chunk header sizes on all memory context types  
Date: 2022-09-01 00:12:20  
Message-ID: [a6ab9367-bcf4-116a-39b7-c9b1afbe8cfc@enterprisedb.com](mailto:a6ab9367-bcf4-116a-39b7-c9b1afbe8cfc@enterprisedb.com)  
Views: [Raw Message](#) | [Whole Thread](#) | [Download mbox](#) | [Resend email](#)  
Thread: [2022-09-01 00:12:20 from Tomas Vondra <tomas\(dot\)vondra\(at\)enterprisedb\(dot\)com>](#)  
Lists: [pgsql-hackers](#)

# Getting Involved

## Start by reviewing patches

- Does the use case make sense?
- Will the proposed change have unintended consequences?
- Does the change work?
- Does the patch follow code style?
- Is it understandable and maintainable?

# Getting Involved

## Submitting a patch

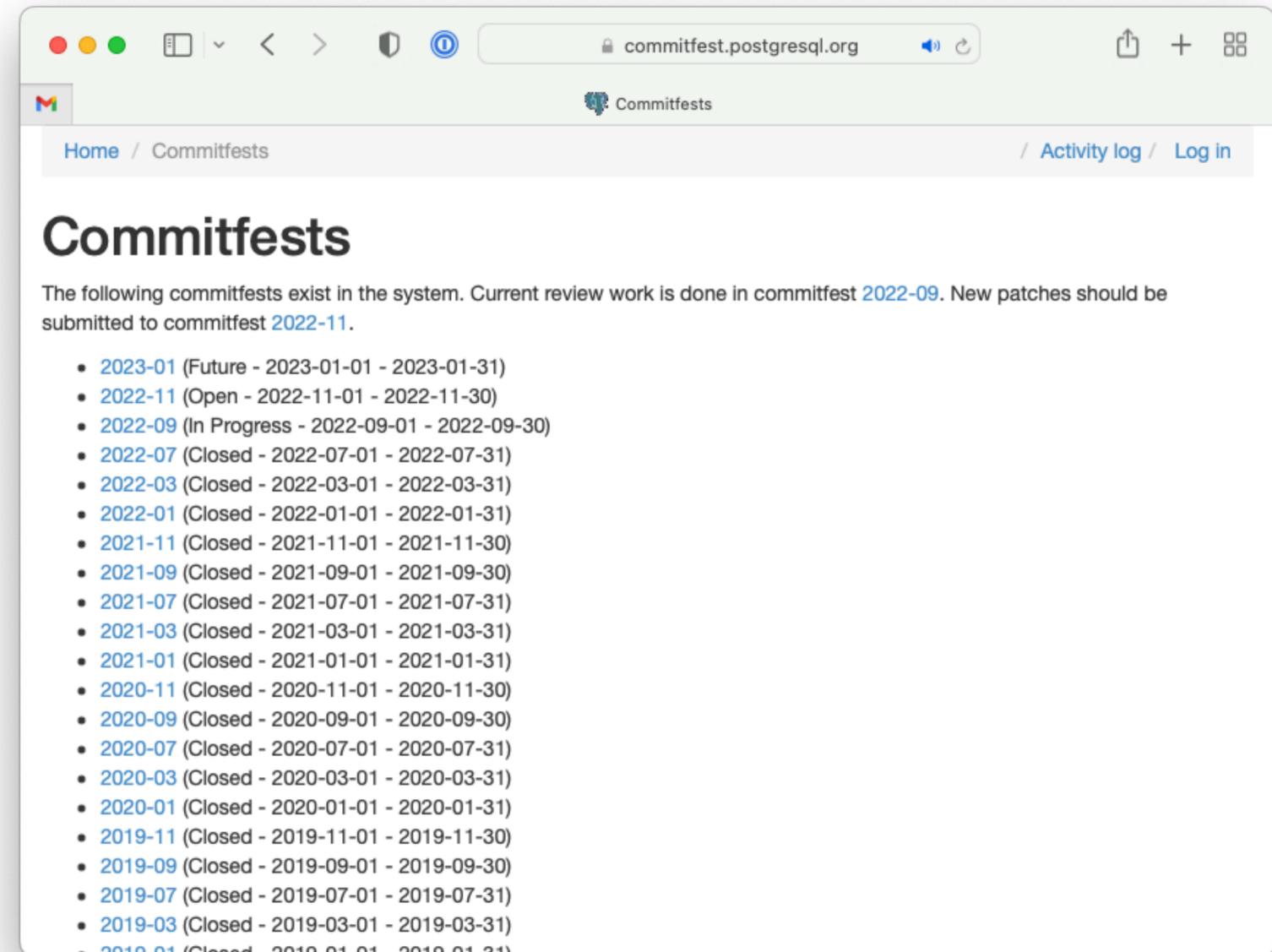
- First search for prior art and discussion
  - If the same approach has been tried before and rejected explain how your patch is different (or reconsider submitting at all)
- Explain the use case you're addressing
  - Sample queries and data are often helpful
  - Show evidence of performance improvement (if applicable)
- Don't make unrelated changes
- For every patch you submit you should review a similarly sized patch.

# CommitFests

- Development is organized into alternating cycles of development and review.
  - After the March CommitFest (at which point a release branch is generally cut) there's usually a multi-month break.
- Each CommitFest is a ~1 month period where *new* development is paused and contributors review existing patches.
  - New patches submitted during a CommitFest will likely be ignored while the CommitFest (review cycle) is active.
- A volunteer CommitFest manager ensures patches in the current cycle are tracked with the right status.

# CommitFests

- The CommitFest (or colloquially “CF”) application [1] tracks patch and CommitFest status.
- New patches are added to the “Open” CommitFest.
- The “In Progress” CommitFest contains the patches currently being reviewed.

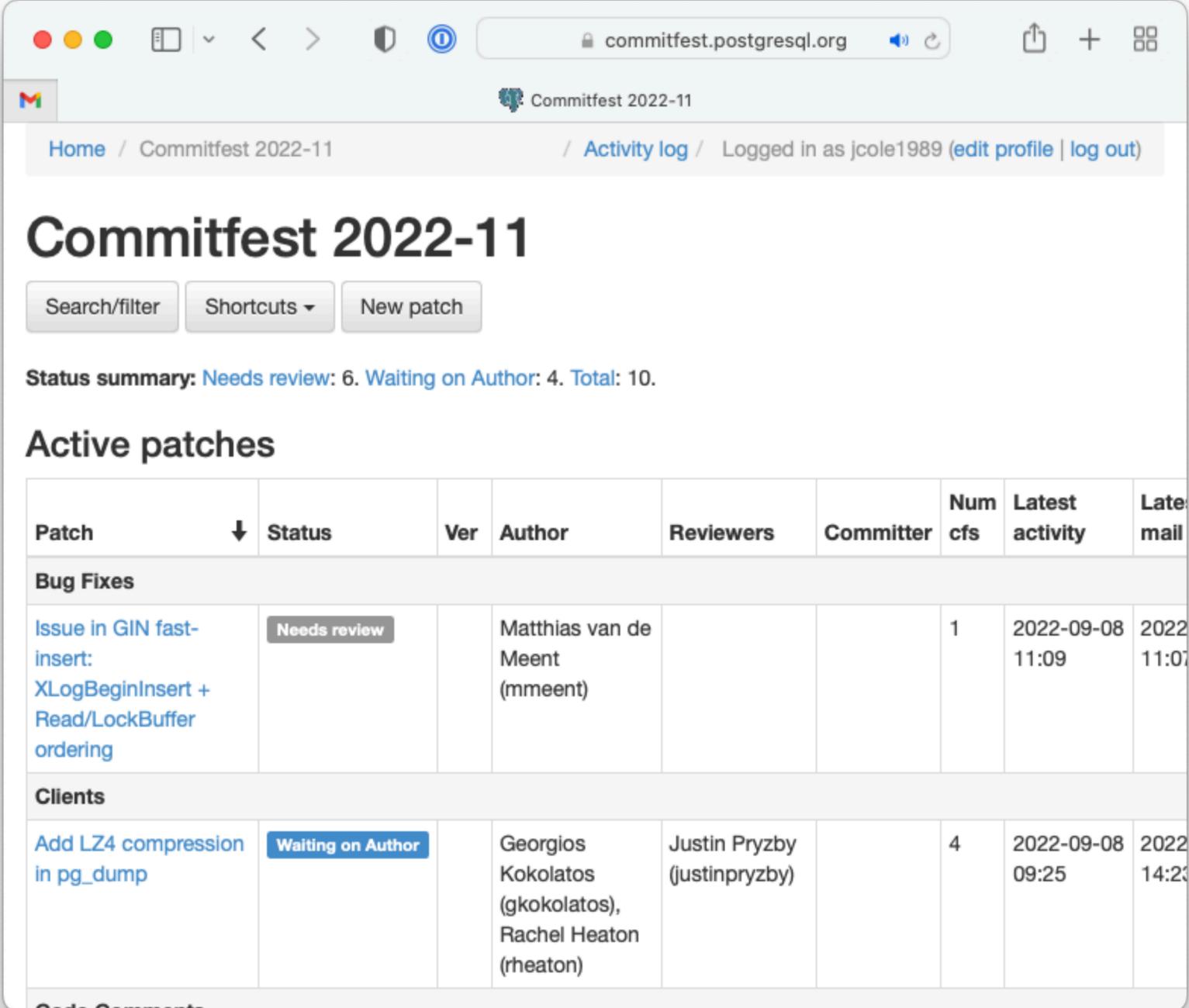


1. <https://commitfest.postgresql.org>

# CommitFests

## Reviewing a patch

You can simply look at recent emails and respond to one you find interesting, or you can look for “Needs Review” patches in the CF app.



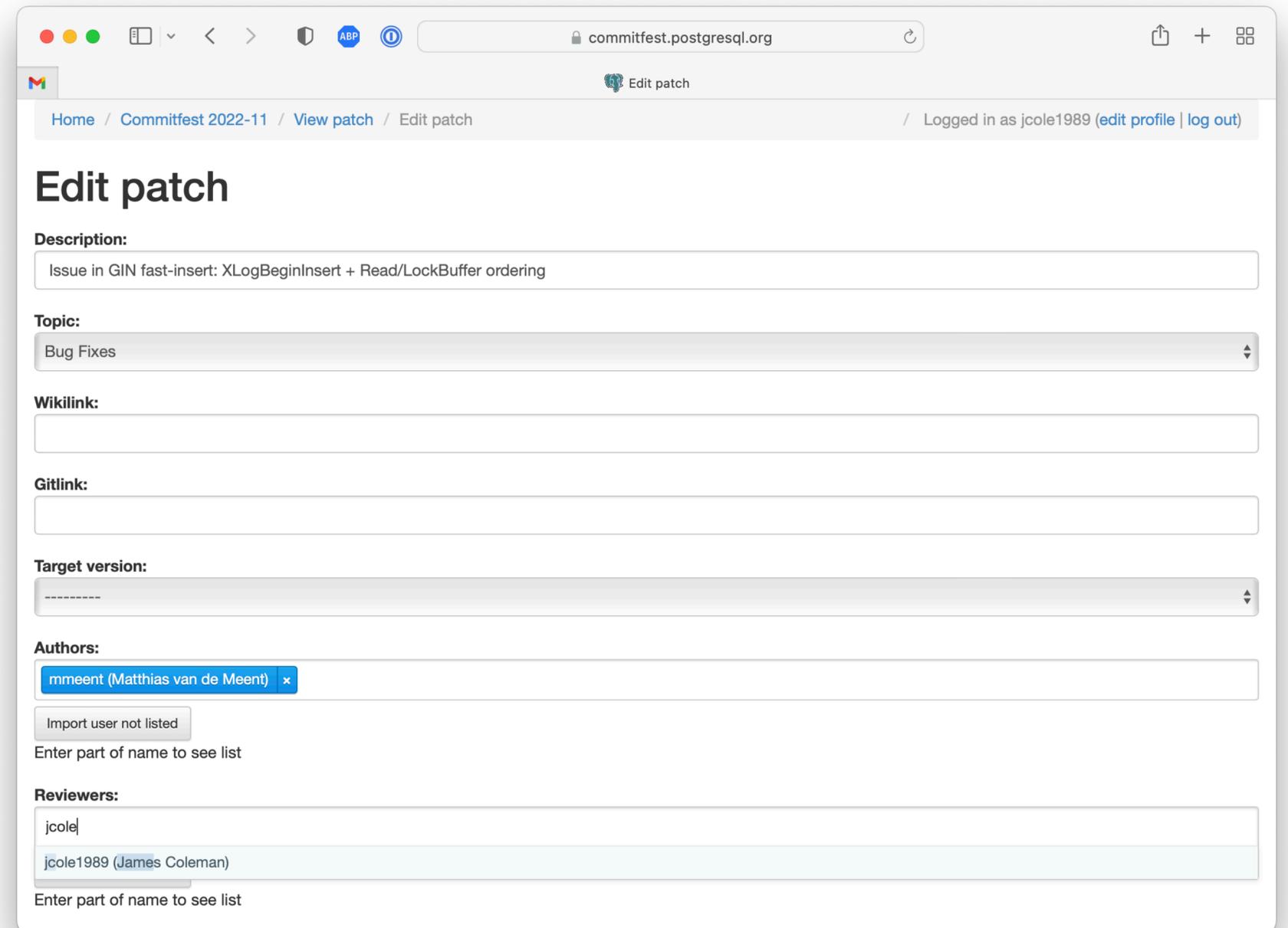
The screenshot shows the web interface for Commitfest 2022-11. The browser address bar shows the URL `commitfest.postgresql.org`. The page title is "Commitfest 2022-11". The user is logged in as `jcole1989`. The page displays a status summary: "Needs review: 6. Waiting on Author: 4. Total: 10." Below this, there is a section for "Active patches" with a table listing patches. The table has columns for Patch, Status, Ver, Author, Reviewers, Committer, Num cfs, Latest activity, and Latest mail. Two patches are visible: one under "Bug Fixes" and one under "Clients".

Patch	Status	Ver	Author	Reviewers	Committer	Num cfs	Latest activity	Latest mail
<b>Bug Fixes</b>								
<a href="#">Issue in GIN fast-insert: XLogBeginInsert + Read/LockBuffer ordering</a>	Needs review		Matthias van de Meent (mmeent)			1	2022-09-08 11:09	2022-09-08 11:07
<b>Clients</b>								
<a href="#">Add LZ4 compression in pg_dump</a>	Waiting on Author		Georgios Kokolatos (gkokolatos), Rachel Heaton (rheaton)	Justin Pryzby (justinpryzby)		4	2022-09-08 09:25	2022-09-08 14:23

# CommitFests

## Reviewing a patch

Either way, if you review a patch you can Edit the patch record in the CF app and record yourself as a reviewer.



The screenshot shows a web browser window with the URL `commitfest.postgresql.org`. The page title is "Edit patch". The breadcrumb navigation is "Home / Commitfest 2022-11 / View patch / Edit patch". The user is logged in as "jcole1989".

The form fields are:

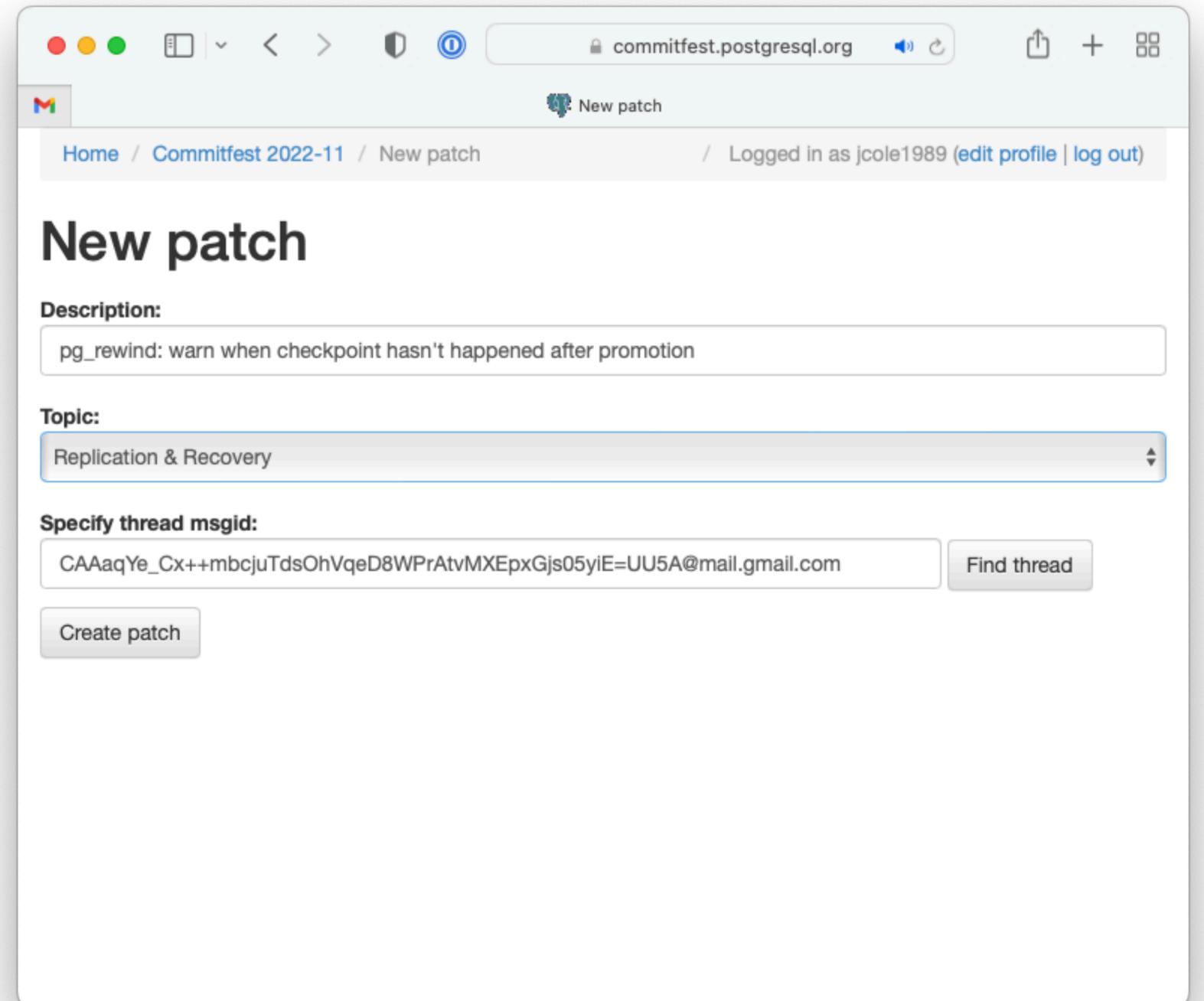
- Description:** Issue in GIN fast-insert: XLogBeginInsert + Read/LockBuffer ordering
- Topic:** Bug Fixes
- Wikilink:** (empty)
- Gitlink:** (empty)
- Target version:** (empty)
- Authors:** mmeent (Matthias van de Meent) x
- Reviewers:** jcole1989 (James Coleman)

Buttons: "Import user not listed", "Enter part of name to see list" (for both authors and reviewers).

# CommitFests

## Adding a new patch

- After sending your email to the mailing list, click “New Patch” in the Open CommitFest.
- Find and attach your message thread (“Find thread” shows the most recent mailing list messages and allows search).



The screenshot shows a web browser window at commitfest.postgresql.org. The page title is "New patch". The breadcrumb navigation is "Home / Commitfest 2022-11 / New patch". The user is logged in as jcole1989. The form contains the following fields:

- Description:** A text input field containing "pg\_rewind: warn when checkpoint hasn't happened after promotion".
- Topic:** A dropdown menu with "Replication & Recovery" selected.
- Specify thread msgid:** A text input field containing "CAAaqYe\_Cx++mbcjuTdsOhVqeD8WPrAtvMXEpxGjs05yiE=UU5A@mail.gmail.com" and a "Find thread" button.
- A "Create patch" button at the bottom.

# Other Resources

- PostgreSQL Wiki – Development information [1]
  - Developer meeting notes
  - Unofficial TODO lists and roadmaps
  - Patch FAQs and checklists
  - Editor and tooling information

1. [https://wiki.postgresql.org/wiki/Development\\_information](https://wiki.postgresql.org/wiki/Development_information)

**Codebase**

# Directory Layout

- `contrib/` - Source for tools, utilities, and extensions that aren't part of the core installation but are nonetheless maintained as part of the main source tree
- `docs/` - SGML source for public documentation
- `src/` - Source for core installation (including tests)

# Directory Layout

## Core source

- Not going to list everything here (see Stephen Frost's talk, linked earlier, for a more in-depth listing), but a few highlights:
  - `src/backend/` - server side of Postgres (\*.c)
  - `src/include/` - server side of Postgres (\*.h)
  - `src/bin/` - front-end tools for Postgres (psql, pg\_\* command line executables, etc.)
  - `src/test/` - regression tests

# Backend Structure

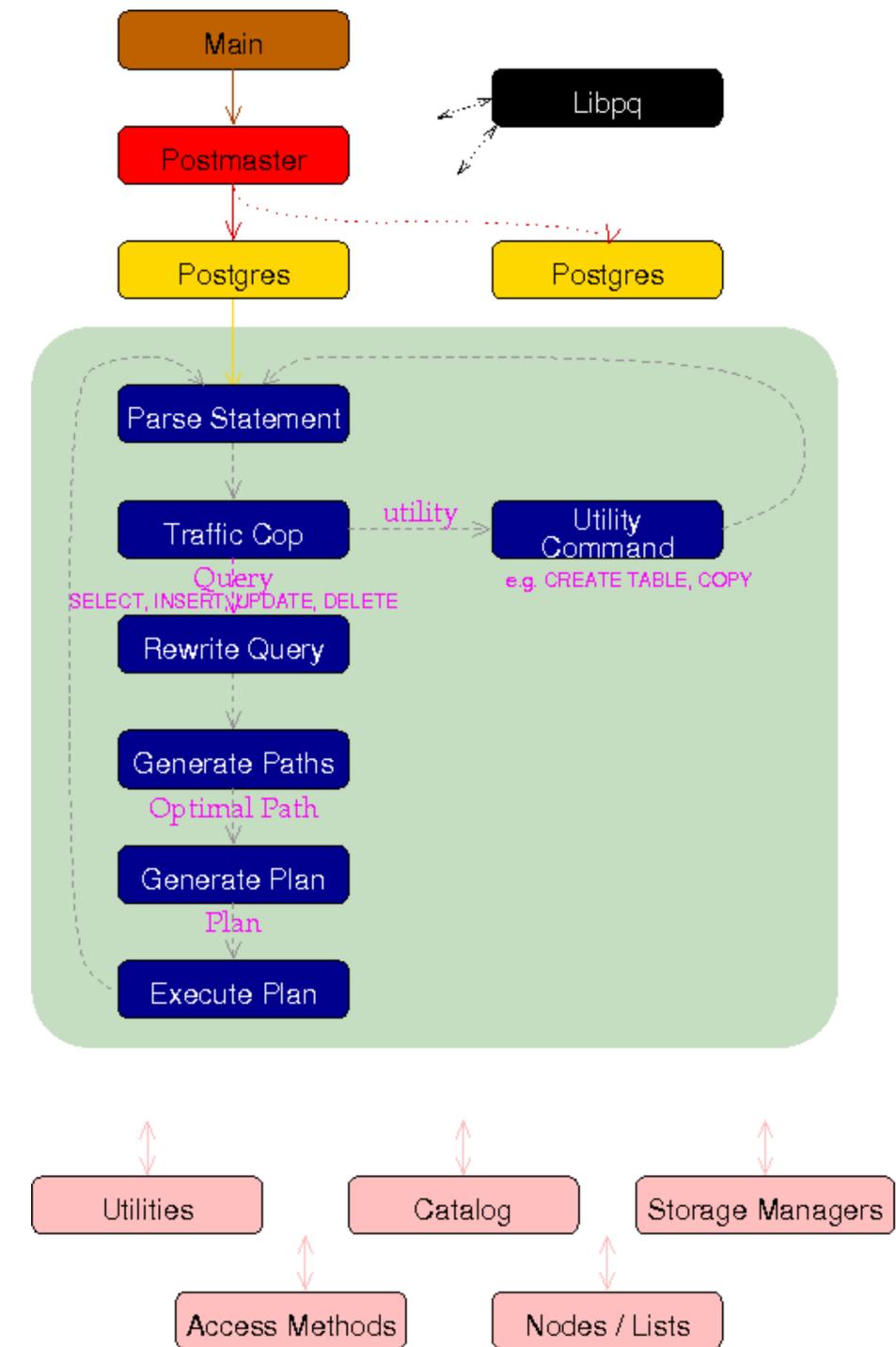
## What component are you working on?

- E.g., query execution includes:
  - Parser
  - Optimizer
  - Executor
- The Postgres source is fairly well organized; each subsystem or component generally has a directory.

# Backend Structure

## How do the components fit together?

- Backend flow chart is in the docs [1]
- Bruce Momjian has a talk “PostgreSQL Internals Through Pictures” referencing this also (and more) [2]
- The Internals of PostgreSQL (online book) [3]



1. <https://www.postgresql.org/developer/backend/>

2. <https://momjian.us/main/presentations/internals.html>

3. <https://www.interdb.jp/pg/>

# Understanding the Source

## README files

- The Postgres source contains a significant number of helpful in-tree documents as README files.
- For example, see the long doc at `src/backend/optimizer/README` that includes data structure information, information on valid JOIN tree construction, and plan generation.

```
298 Optimizer Functions
299 -----
300
301 The primary entry point is planner().
302
303 planner()
304 set up for recursive handling of subqueries
305 -subquery_planner()
306 pull up sublinks and subqueries from rangetable, if possible
307 canonicalize qual
308     Attempt to simplify WHERE clause to the most useful form; this includes
309     flattening nested AND/ORs and detecting clauses that are duplicated in
310     different branches of an OR.
311 simplify constant expressions
312 process sublinks
313 convert Vars of outer query levels into Params
314 --grouping_planner()
315 preprocess target list for non-SELECT queries
316 handle UNION/INTERSECT/EXCEPT, GROUP BY, HAVING, aggregates,
317 ORDER BY, DISTINCT, LIMIT
318 ---query_planner()
319 make list of base relations used in query
320 split up the qual into restrictions (a=1) and joins (b=c)
321 find qual clauses that enable merge and hash joins
```

# Understanding the Source

## Comments

- The Postgres source is heavily commented.
- Your code should generally include comment headers for each function.
- Inline to code you should explain reasoning for why assumptions hold true, what you're trying to accomplish, etc.

```
1393  /*
1394  * ScalarArrayOpExpr is a special case. Note that we'd only reach here
1395  * with a ScalarArrayOpExpr clause if we failed to deconstruct it into an
1396  * AND or OR tree, as for example if it has too many array elements.
1397  */
1398  if (IsA(clause, ScalarArrayOpExpr))
1399  {
1400      ScalarArrayOpExpr *saop = (ScalarArrayOpExpr *) clause;
1401      Node *scalarnode = (Node *) linitial(saop->args);
1402      Node *arraynode = (Node *) lsecond(saop->args);
1403
1404      /*
1405      * If we can prove the scalar input to be null, and the operator is
1406      * strict, then the SAOP result has to be null --- unless the array is
1407      * empty. For an empty array, we'd get either false (for ANY) or true
1408      * (for ALL). So if allow_false = true then the proof succeeds anyway
1409      * for the ANY case; otherwise we can only make the proof if we can
1410      * prove the array non-empty.
1411      */
1412      if (clause_is_strict_for(scalarnode, subexpr, false) &&
1413          op_strict(saop->opno))
1414      {
1415          int nelems = 0;
1416
1417          if (allow_false && saop->useOr)
1418              return true; /* can succeed even if array is empty */
1419      }
```

# Style

- Use proper project style to avoid unnecessary frustration with your patch! [1]
  - Tabs, not spaces, displayed as 4 columns per tab stop.
  - Use a new line for opening braces; no braces around single statements.
  - 80 character column limit.
- Be sure to follow the style of the surrounding code:
  - E.g., there are unfortunately lots of different variable naming styles (camel-case, underscores, etc.); match the context as much as possible.

1. <https://www.postgresql.org/docs/devel/source-format.html>

# Docs

- The `docs/` top level directory contains the SGML/XML source for public documentation.
- The markup uses DocBook [1].
- Note: DocBook source formatting style differs from the C source code.
  - Single space indenting.

1. <https://www.postgresql.org/docs/current/docguide-docbook.html>

# Built-in Facilities

# Memory Management

## Memory Contexts

- Postgres tracks memory usage as part of nested memory contexts.
- `TopMemoryContext` exists for the lifetime of a backend
- New contexts are created for each query, operations within a query (e.g., a sort), sometimes per tuple, etc.
- `CurrentMemoryContext` is used for new allocations.

# Memory Management

## Managing Contexts

- There are multiple memory allocators available:
  - `AllocSet` [1] is the standard allocator; maintains free lists in larger blocks of memory.
  - `Generation` [2] is useful for limiting underlying `malloc/free` calls when memory is chunks are used in a roughly FIFO manner (e.g., a queue).
  - `Slab` [3] is useful when “large numbers of equally-sized objects are allocated (and freed).”

1.src/backend/utis/mmgr/aset.c

2.src/backend/utis/mmgr/generation.c

3.src/backend/utis/mmgr/slab.c

# Memory Management

## Managing Contexts

- (If needed) create a new context with `{AllocSet, Generation, Slab}ContextCreate(...)`
- Switch contexts with `MemoryContextSwitchTo(context)`; make sure to switch back. Common pattern looks like:

```
MemoryContext oldcxt;  
oldcxt = MemoryContextSwitchTo(some_cxt);
```

```
<do work>
```

```
MemoryContextSwitchTo(oldcxt);
```

# Memory Management

## Managing Allocations

- `palloc(Size size)`
- `palloc0(Size size)`
- `palloc_extended(Size size, int flags)`
- `realloc(void *pointer, Size size)`
- `pfree(void *pointer)`
  - Note: sometimes you won't need to explicitly free memory because the entire context's allocations are cleaned up with `MemoryContextReset` or `MemoryContextDelete` (and friends).

# Logging and Error Handling

- Both are handled through the same infrastructure:

```
ereport(<level>  
        errcode(ERRCODE_...), # Optional  
        errmsg(...),  
        ...) # Optional fields; e.g. errdetail(...) and errhint(...)
```

# Logging and Error Handling

## Log levels (non-error case)

- `DEBUG{5,4,3,2,1}`
- `LOG`: Operational messages sent to server log by default.
- `INFO`: Explicitly requested by user (e.g., `VERBOSE`); sent to client and not server log by default.
- `NOTICE`: User-targeted helpful, expected messages; sent to client and not server log by default.
- `WARNING`: Like `NOTICE`, but unexpected messages.

# Logging and Error Handling

## Log levels (error case)

- ERROR:
  - Abort current transaction.
  - Doesn't return to caller.
  - Cleans up memory, etc.
- FATAL: Abort current process.
- PANIC: Shutdown everything.

# Data Structures

## Lists

- `List [1]`: simple, expansible array implementation; empty list is `NIL`
- `slist_head, dlist_head [2]`: Single and doubly linked lists.

# Data Structures

## Hashtables

- Simplehash [1]
  - Templated (by way of macros) specialized implementations for user types (improves speed and memory usage at cost of complex setup and increased binary size)
  - Open-addressing (good for CPU cache behavior)
- Dynahash [2] chained hashtable
  - Shared memory (fixed size at startup) or backend-local.
  - Partitionable (improves shared memory access locking performance)
  - Guarantees stable pointers (hash conflicts don't result in moving entries, thus more performant for large keys)

1.src/include/lib/simplehash.h

2.src/backend/utils/hash/dynahash.c

# Data Structures

## Hashtables

- `dshash_hash` [1]
  - Concurrent
  - Dynamic shared memory

# Data Structures

## Other

- `binaryheap` [1]: full/balanced binary tree
- `Bitmapset` [2]: set of non-negative integers (usually max value is low)
- `bloom_filter` [3]: Space-efficient set membership testing
- `IntegerSet` [4]: efficient large integer set; uses a B-tree internally with nearby values stored in a packed representation

1.`src/include/lib/binaryheap.h`

2.`src/include/nodes/bitmapset.h`

3.`src/include/lib/bloomfilter.h`

4.`src/include/lib/integerset.h, src/backend/lib/integerset.c`

# Data Structures

## Other

- `pairingheap` [1]
- `RBTtree` [2]
- `StringInfo` [3]: Extensible string buffer type (up to 1GB)

1.`src/include/lib/pairingheap.h`

2.`src/include/lib/rbtree.h`

3.`src/include/lib/stringinfo.h`

# Algorithms

- Binary search: implemented inline in several places
- Bipartite matching [1]
- HyperLogLog [2]: cardinality (unique values) estimation
- Knapsack [3]
- Uniquing arrays [4]
- Templated (macros) sorting implementations [5]

1.src/include/lib/bipartite\_match.h

2.src/include/lib/hyperloglog.h

3.src/include/lib/knapsack.h, src/backend/lib/knapsack.c

4.src/include/lib/unique.h

5.src/include/lib/sort\_template.h

# GUCs (Grand Unified Configuration)

## Runtime configuration

- Configured in conf files or set at runtime with the familiar SET <config\_name> ... syntax.
- Whether runtime configuration is needed permanently for your feature is necessary or not adding a GUC is often useful during development to switch a feature on/off without rebuilding.
- Can have default and upper and lower bounds (if applicable).
- Optional hooks for showing, setting, and checking valid values.

# GUCs (Grand Unified Configuration)

## Adding a new GUC (simplified)

- Declare a global variable (`bool`, `int`, `double`, or `char*`) in your file.
- Add an entry to the appropriate `ConfigureNames{Bool,Int,Real,String}` table in `src/backend/utils/misc/guc_tables.c`.
- For more details, see the “TO ADD AN OPTION” section in `src/backend/utils/misc/guc_tables.c`.

# Tooling

**Working on a patch**

# Environment

## Editor, etc.

- I'm going to use my preferred setup [1] in the following demo.
  - Vim: for editing
  - Tmux: working in a terminal with multiple windows and panes
  - Some custom scripting [2]
- My setup isn't the only way (but if you're using emacs...you're wrong 🤪).
- Sample editor configuration files can be found in-tree at `src/tools/editors/`

1.<https://github.com/jcoleman/machine-setup>

2.<https://github.com/jcoleman/postgres-dev-tools>

# **Terminal Demo**

Applying a patch, building, and debugging

# Debugging

- `elog(WARNING, "...")` for good old fashioned print debugging (and it goes to the client if you're working in query execution!)
- If you encounter a segfault or other crash be sure you've rebuilt (cleanly) and re-initialized your data directory since your last pull/rebase.
- `gdb` or `lldb` for an interactive debugger.
- `rr [1]` can "record" execution runs for replay.

1.[https://wiki.postgresql.org/wiki/Getting\\_a\\_stack\\_trace\\_of\\_a\\_running\\_PostgreSQL\\_backend\\_on\\_Linux/BSD#Recording\\_Postgres\\_using\\_rr\\_Record\\_and\\_Replay\\_Framework](https://wiki.postgresql.org/wiki/Getting_a_stack_trace_of_a_running_PostgreSQL_backend_on_Linux/BSD#Recording_Postgres_using_rr_Record_and_Replay_Framework)

# Testing

## Running the tests

- Run core regression tests with `make check`
  - Can run against local install with `make installcheck`
- Run all tests with `make check-world`
  - Can run against local install with `make installcheck-world`

# Testing

## Regression tests

- Found in `src/test/regress/{sql,expected,results}/`
- Consist of SQL files containing “tests” and the expected output from running those files with `psql`
- Logs found in `src/test/regress/log/postmaster.log`
- Not isolated/may reference previous files (can make pulling a test out difficult)
- Default to parallel groups; can run serially by setting `MAX_CONNECTIONS=1`

1.<https://www.postgresql.org/docs/current/regress.html>

2.[https://wiki.postgresql.org/wiki/Regression\\_test\\_authoring](https://wiki.postgresql.org/wiki/Regression_test_authoring)

# Testing

## TAP tests

- Found in \*.pl files in a t/ subdirectories (e.g., of the source for various binaries like src/bin/pg\_rewind/t/)
- Run individual test files (while in e.g. src/bin/pg\_rewind/) with `make check PROVE_TESTS=t/010_no_checkpoint_after_promotion.pl`
- Logs found in tmp\_check/log subdirectory

1.<https://www.postgresql.org/docs/current/regress-tap.html>

2.<https://pgtap.org>

# Testing

## CI

- Official repository is tested by the “buildfarm” made up of many user-run machines referred to as “animals” (each with a fun name to boot).
- You can configure CirrusCI on your personal fork of the repo on GitHub:
  - Supports various platforms so you can test against system types you don’t have personal access to/different from your development environment.
  - See `src/tools/ci/README` for details.

# Submitting a Patch

- Break your change into a coherent set of patches (if necessary); for example:
  - You might have a precursor patch that is a clean refactor and then a second patch to make your actual change.
  - You might have a second patch that's useful for debugging (e.g., adding GUCs you won't actually want to keep around).
- Write message for mailing list (and commit messages) including:
  - Motivation for change
  - How the change works
  - What you expect the outcome to be

# Submitting a Patch

- Ensure code is formatted properly:
  - `pgindent` [1] (found in source tree) is the canonical tool for formatting source.
  - Requires some tools to be installed; more information found at [2]
- Patch filenames should include the version of the patch and (if multiple files) the index of the patch in the patch series.
  - `git format-patch -v<n> master`

1.<https://github.com/postgres/postgres/tree/master/src/tools/pgindent>

2.<https://www.interdb.jp/blog/pgsql/pgindent/>

**Q/A**