EDB
Postgres® for the AI Generation

Committer Review: An Exercise in Paranoia

Robert Haas
VP, Chief Architect, Database Servers
2025.pgconf.dev

Agenda

- Committer Mindset
- Reviewing Patches

EDB
Postgres® for the AI Generation

Committer Mindset

What are those people thinking? And why won't they just commit my patch already?

Committer Responsibility

- If you commit someone's patch, *you are responsible for whatever goes wrong*.
- There are no excuses.
- If you break the buildfarm, you must fix it immediately.
- Even years later, you may need to:
  - defend the design
  - fix bugs
  - maintain the code (e.g. JIT depends on LLVM, so LLVM changes require adjustments to JIT)
- It is unlikely you'll get much help from the patch author or any of the reviewers.

Mistakes Are Easy To Make

- Even our most experienced and talented committers regularly introduce serious bugs (and so do I).
- Data corruption bugs and server crashes make it into the source tree fairly often, and sometimes aren't caught before release.
- Smaller mistakes are even more common, and post-commit fixups are very common.

```
83ccc85859f pg_overexplain: Use PG_MODULE_MAGIC_EXT.
9f0c36aea0f pg_overexplain: Call previous hooks as appropriate.
081ec08e6a1 pg_overexplain: Filter out actual row count from test result.
47a1f076a7c pg_overexplain: SET jit=off when running tests.
de65c4dade0 Fix oversights in commit 8d5ceb113e3f7ddb627bd40b26438a9d2fa05512
8d5ceb113e3 pg_overexplain: Additional EXPLAIN options for debugging.
```

```
commit beb850e1d873f8920a78b9b9ee27e9f87c95592f
Author: Robert Haas <rhaas@postgresql.org>
Date:   Fri Sep 14 09:35:07 2012 -0400

    Properly set relpersistence for fake relcache entries.

    This can result in buffers failing to be properly flushed at
    checkpoint time, leading to data loss.

    Report, diagnosis, and patch by Jeff Davis.
```

Committer Experience

- Being a committer is a great privilege in many ways.
- It’s easy to imagine that this is how committers feel when they type `git push`

Committer Fear

- But the committers I've talked to feel more like this: committing patches is *stressful*.
- They engage in painstaking, exhaustive verification of a patch set before pushing.
- They're often as afraid to push their own patches as anyone else's.
- Then things still go wrong and they have to scramble.
- There's extra work, there's public embarrassment, and maybe even loss of commit privileges.

EDB
Postgres® for the AI Generation

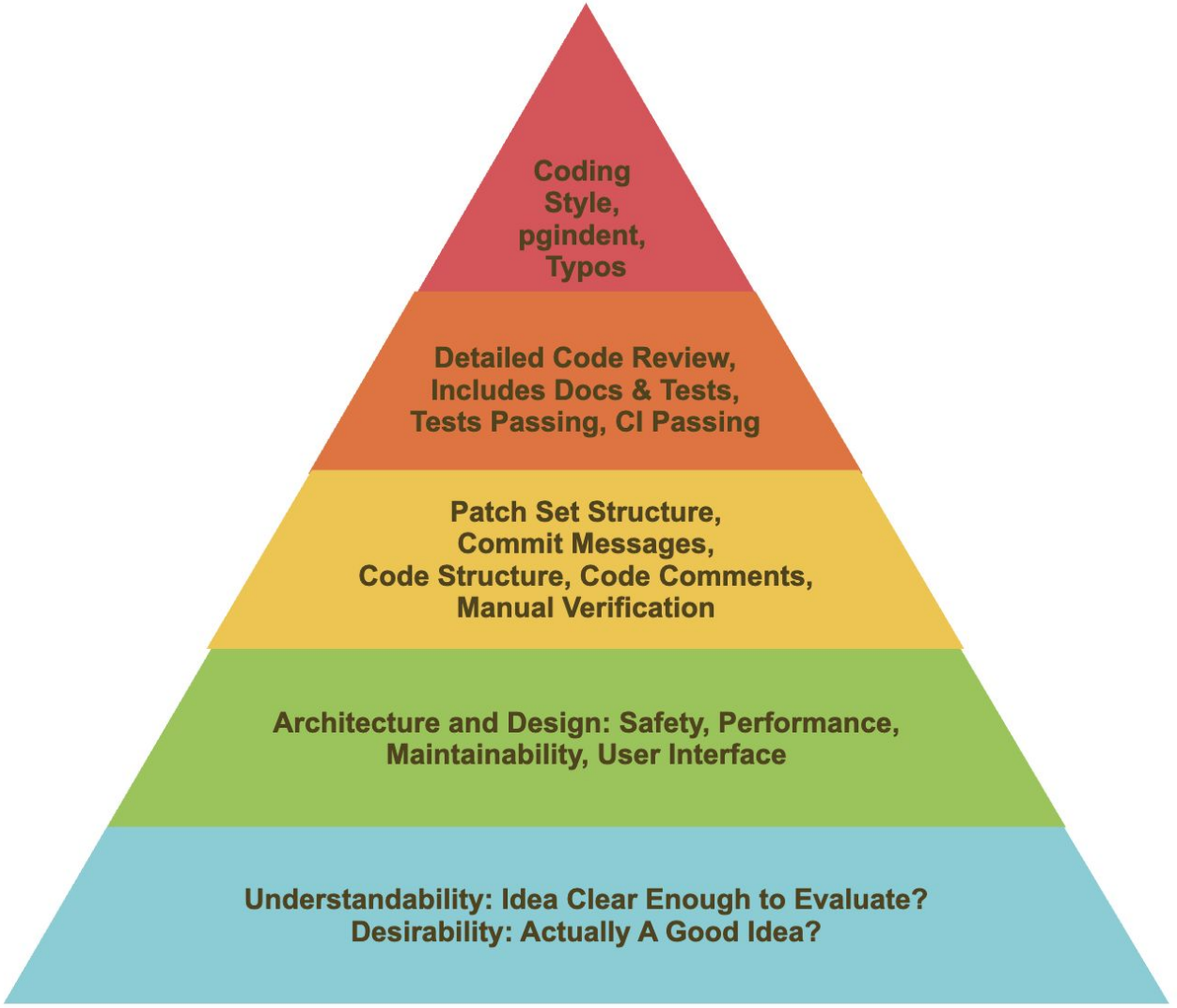
Reviewing Patches

What is the committer going to be checking? And what should I check as an author or reviewer?

Reviewing Patches: Key Principles

- *Start With The Big Picture*. If the design of the patch is incorrect in some fundamental way, it doesn't really matter whether there are typos.
- *Consistency Is Critical*. Deviations from existing practice need to be well-justified.
- *Be a Pessimist*. Nobody submits a patch unless they think it's good; being a good reviewer requires noticing what the author missed.
- *Apply a Spirit of Maintainership*. It's not about getting one particular patch accepted, it's about what's good for the project overall.

Patch Review Hierarchy of Needs



Start With The Big Picture: An Aspirational Goal

- I often find that my first pass through a complex patch finds a lot of small stuff because it's easier to find - and because I don't yet really understand the patch.
- Sometimes I need to ask for improvements to comments or commit messages in order to be able to properly understand the patch so that I can review it.

Reviewing For Consistency - Why?

- Refutes political objections - “we’ve done X before so it can’t be wrong to do it again”
- Avoids bugs - “the original works so my copy will work too”
- Makes the code more readable and maintainable - “oh, this looks familiar”
- Makes the user experience more consistent - “wait, it’s DROP TABLE but REMOVE FUNCTION?”

Reviewing For Consistency - When?

- Applies to both high-level review and detailed review.
- When we're reviewing the design, we can ask whether this design is similar to other designs, and whether there are good reasons for any differences.
- We can ask the same questions during line-by-line code review.
- PostgreSQL's codebase is full of "coding micro-practices" where we (almost) always do certain things in the same way; deviations are discouraged.

Coding Micro-Practices: Example

```
extern char **
GetDatabaseNames(unsigned n, Oid *oids)
{
    char       **names = palloc0(n * sizeof(char *));
    unsigned    i;
    HeapTuple   h;
    Form_pg_database dbstruct;

    for (i = 0; i < n; ++i)
    {
        h = SearchSysCache(DATABASEOID, oids[i], 0, 0, 0);
        if (h == NULL)
            elog(ERROR, "pg_database entry missing for OID %u", oids[i]);
        dbstruct = GETSTRUCT(h);
        names[i] = NameStr(dbstruct->datname);
        ReleaseSysCache(h);
    }

    return names;
}
```

Coding Micro-Practices: Problems

```
extern char **
GetDatabaseNames(unsigned n, Oid *oids)
{
    char       **names = palloc0(n * sizeof(char *));
    unsigned    i;
    HeapTuple   h;
    Form_pg_database dbstruct;

    for (i = 0; i < n; ++i)
    {
        h = SearchSysCache(DATABASEOID, oids[i], 0, 0, 0);
        if (h == NULL)
            elog(ERROR, "pg_database entry missing for OID %u", oids[i]);
        dbstruct = GETSTRUCT(h);
        names[i] = NameStr(dbstruct->datname);
        ReleaseSysCache(h);
    }

    return names;
}
```

Be a Pessimist

- Most (certainly not all) of my failures as a reviewer are the result of insufficient pessimism.
- The author might have missed:
  - a critical design flaw
  - some necessary code modifications
  - a functional or performance regression
  - a discrepancy between the comments and the code

Pessimism: Searching for Oversights

- For example, if we're changing something related to identifier `XYZ`, try `git grep XYZ` to find all of the other places that need to be changed.
- Or search for a similar identifier that might serve as precedent, e.g. if adding `ACL_TRUNCATE`, try `git grep ACL_DELETE`
- Think about "pairings" e.g. if we're allocating memory or creating a file or setting a global variable, consider whether we need to take any action to free the memory or delete the file unset the global variable.
- Consider system-wide consequences e.g. if you add `CREATE STRAWBERRY`, then `pg_dump` will need to dump-and-restore strawberries, psql might need `\dstrawberry` etc.

Pessimism: Functional or Performance Regressions

- Functional regressions: If this patch had a bug, where might it be?
- Performance regressions: If this patch slows something down, under what circumstances might it occur?
- Hands-on testing and benchmarking can be very helpful, but reading the code is usually necessary to target the activity.
- Even then, it's easy to think that you're testing or benchmarking the right thing when in fact you are not.
  - Consider adding temporary debugging code or using a debugger or profiler to make absolutely sure you are hitting the right code path.

The Spirit of Maintainership

- Patches must promote the good of the project, not any one person.
- People are often think that their patch is worth a cost such as:
  - a 3% performance regression
  - a user interface wart
  - an annual code maintenance task
- But those costs add up quickly, so we need to be very judicious about incurring them.

EDB
Postgres® for the AI Generation

Thank you!

Any questions?