

How to Get Your PostgreSQL Patch Accepted

BRUCE MOMJIAN, ENTERPRISEDB
TOM LANE, RED HAT



January, 2012

Developing a patch for the PostgreSQL project is a fairly complex process, and success is not guaranteed. This talk will suggest many ways to improve your chances of submitting a successful patch to the PostgreSQL community.

Creative Commons Attribution License

<http://momjian.us/presentations>
How to Get Your PostgreSQL, Patch Accepted 1/29

Know the Community Values: Is it This?



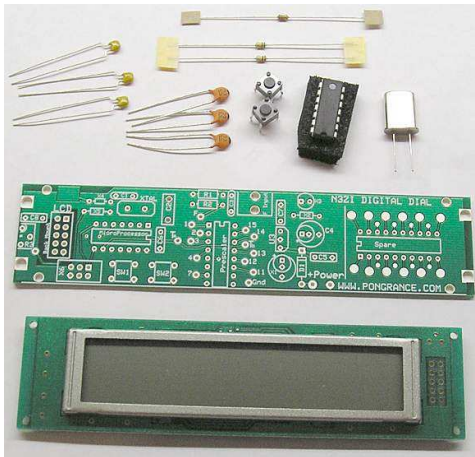
<http://www.friedmanarchives.com>

Or This?

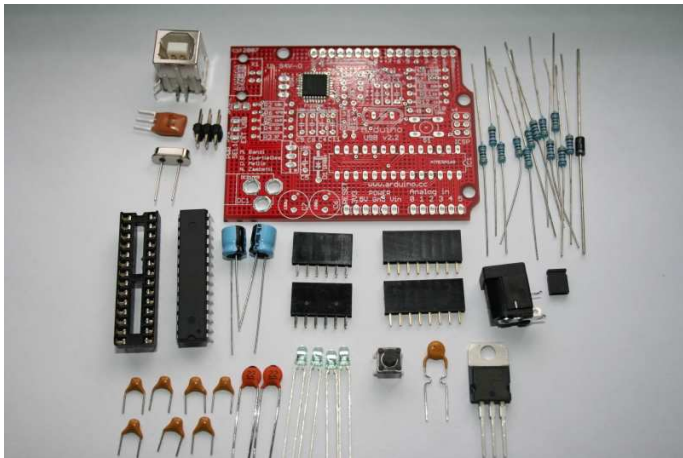


The PostgreSQL codebase is 20+ years old, and is built expecting another twenty. Code designed for a long life-cycle has more stringent requirements.

Start Simple



Gain Experience Before Attempting Complex Tasks



Don't Start Here



Karoly Arvai/Reuters

Everyone Is Watching



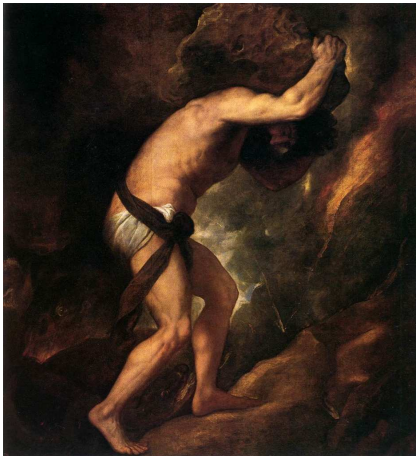
Coding in public can be uncomfortable.

Be Prepared to be Humbled



MXC

Sometimes You Have to Do it Over



Most patches take several attempts.

Get Agreement



Don't Isolate Yourself

Design the Interface First

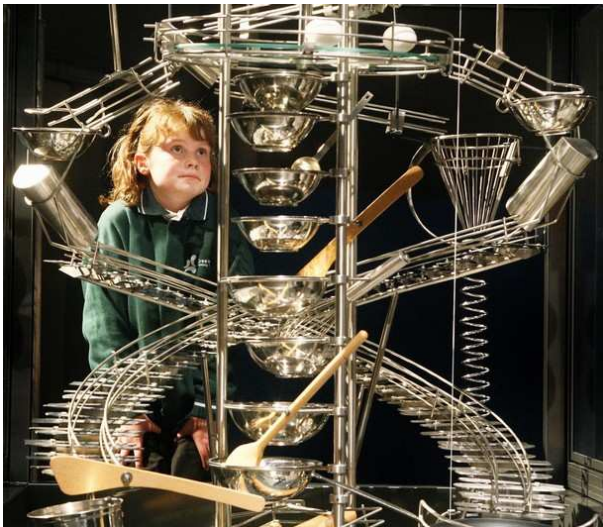


Define the new behavior, and get community agreement.

Interface Details

- ▶ Will the user interact with this new feature? If so, how?
- ▶ What is the syntax?
- ▶ What is the exact semantics/behavior?
- ▶ Are there any backward compatibility issues?
- ▶ Get community buy-in at this level of detail before you start coding.

Implementation Mechanics Second



How will the new behavior be implemented?

Implementation Details

- ▶ What subsystems and C files will be modified?
- ▶ How does it interact with other database features?
- ▶ How will error conditions be handled?
- ▶ Are there any system catalog changes?
- ▶ Are there any dump/restore issues? If so, fix `pg_dump`.
- ▶ Does `psql` need to be modified?

Formatting Matters

```
main(int c,char**v){return!m(v[1],v[2]);}m(char*s,char*t)
{return*t-42?*s?63==*t|*s==*t&& m(s+1,t+1):!*t:m(s,t+1)||
*s&& m(s+1,t);}

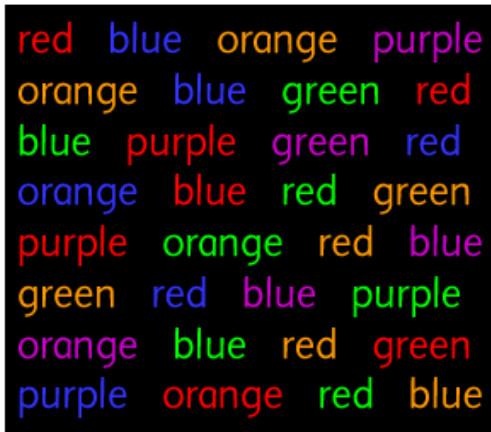
```

The International Obfuscated C Code Contest

Formatting Details

- ▶ Four-space tabs
- ▶ BSD brace style
- ▶ Pgindependent cures some ills, but not all
- ▶ Mimic surrounding code

Naming Matters



red	blue	orange	purple				
orange	blue	green	red				
blue	purple	green	red				
orange	blue	red	green				
purple	orange	red	blue				
green	red	blue	purple				
orange	blue	red	green				
purple	orange	red	blue				

Naming Details

- ▶ `thisStyleIsOkay`
- ▶ `this_is_okay_too`
- ▶ `besttoavoidthis`
- ▶ `MimicAdjacentCode`
- ▶ `szNotHungarian`

Match Your Surroundings



Match Details

- ▶ Match the style of the surrounding code, even if it differs from other areas
- ▶ Don't use `#ifdef`'s to enable your changes
- ▶ Comments are for clarification, not for delineating your code from its surroundings
- ▶ Do not add your name to the code (the release notes will immortalize your contribution)

Who Needs Comments?

```
for (i = 0; i < len; i++)
{
    char        ch = text[i];

    if (ch >= 'A' && ch <= 'Z')
        ch += 'a' - 'A';
    word[i] = ch;
}
word[len] = '\0';

low = &ScanKeywords[0];
high = LastScanKeyword - 1;
while (low <= high)
{
    const ScanKeyword *middle;
    int                difference;

    middle = low + (high - low) / 2;
    difference = strcmp(middle->name, word);
    if (difference == 0)
        return middle;
    else if (difference < 0)
        low = middle + 1;
    else
        high = middle - 1;
}
```

We Need Comments

```
/*
 * Apply an ASCII-only downcasing. We must not use tolower() since it may
 * produce the wrong translation in some locales (eg, Turkish).
 */
for (i = 0; i < len; i++)
{
    char        ch = text[i];

    if (ch >= 'A' && ch <= 'Z')
        ch += 'a' - 'A';
    word[i] = ch;
}
word[len] = '\0';

/*
 * Now do a binary search using plain strcmp() comparison.
 */
low = &ScanKeywords[0];
high = LastScanKeyword - 1;
while (low <= high)
{
    const ScanKeyword *middle;
    int                difference;

    middle = low + (high - low) / 2;
    difference = strcmp(middle->name, word);
    if (difference == 0)
        return middle;
    else if (difference < 0)
        low = middle + 1;
    else
        high = middle - 1;
}
```

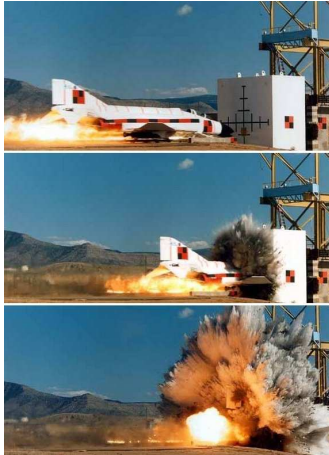
Sometimes A Long Comment Is Necessary

```
/*
 * Replace correlation vars (uplevel vars) with Params.
 *
 * Uplevel aggregates are replaced, too.
 *
 * Note: it is critical that this runs immediately after SS_process_sublinks.
 * Since we do not recurse into the arguments of uplevel aggregates, they will
 * get copied to the appropriate subplan args list in the parent query with
 * uplevel vars not replaced by Params, but only adjusted in level (see
 * replace_outer_agg). That's exactly what we want for the vars of the parent
 * level --- but if an aggregate's argument contains any further-up variables,
 * they have to be replaced with Params in their turn. That will happen when
 * the parent level runs SS_replace_correlation_vars. Therefore it must do
 * so after expanding its sublinks to subplans. And we don't want any steps
 * in between, else those steps would never get applied to the aggregate
 * argument expressions, either in the parent or the child level.
 *
 * Another fairly tricky thing going on here is the handling of SubLinks in
 * the arguments of uplevel aggregates. Those are not touched inside the
 * intermediate query level, either. Instead, SS_process_sublinks recurses
 * on them after copying the Aggref expression into the parent plan level
 * (this is actually taken care of in build_subplan).
 */
Node *
SS_replace_correlation_vars(PlannerInfo *root, Node *expr)
{
    /* No setup needed for tree walk, so away we go */
    return replace_correlation_vars_mutator(expr, root);
}
```

Style Details

- ▶ Proper naming can reduce need for comments
- ▶ Comment anything that could be unclear
- ▶ Use `/*-----` comment blocks if you want to preserve line breaks in comments
- ▶ Use ANSI C (C89), not C99
 - ▶ No `//` comments
 - ▶ Variable declarations only at the top of code blocks (`{}`)

The Importance of Testing



Because it is best to test without passengers.

Testing Details

- ▶ Do existing regression tests pass? Are there intentional changes?
- ▶ Are new regression tests needed?
- ▶ Does initdb work?
- ▶ Are there any nonportable constructs? The buildfarm will quickly report them once the code is committed to CVS, which can be embarrassing.

Documentation Matters



Documentation Details

- ▶ Are new commands or new options supported? Add/update reference pages.
- ▶ Do administrators need to know about this change? Add to administration section.
- ▶ Is this a new function? Add it to the functions documentation.
- ▶ Is there a documentation section that covers this behavior? Does it need adjustment?

Conclusion



"The electric light has caused me the greatest amount of study and has required the most elaborate experiments," Thomas Edison wrote. "I was never myself discouraged, or inclined to be hopeless of success. I cannot say the same for all my associates."

"Genius is one percent inspiration and ninety-nine percent perspiration."

<http://momjian.us/presentations>

Thomas Edison