Against Names

Aug 12 2024

There's an old saying:

> *"There are only two hard things in Computer Science: cache invalidation and naming things.*
>
> *— Phil Karlton"*

I also appreciate the joke version that adds "and off by one errors."

Lately, I've been thinking about this saying, combined with another old joke:

> *""The patient says, "Doctor, it hurts when I do this." The doctor says, "Then don't do that!"*
>
> *— Henny Youngman"*

Specifically, if naming things is so hard... why do we insist on doing it all the time?

Now, I am not actually claiming we should stop giving things names. But I have had at least two situations recently where I previously found names to be kinda critical, and then I changed to systems which didn't use names, and I think it improved the situation.

One of the most famous examples of not giving something a name, lambdas/closures, took some time to catch on. But many folks already recognize that naming every single function isn't always neccesary. I wonder if there are more circumstances where I've been naming things where I didn't actually have to.

Anyway, here's my two recent examples:

Nameless Branches

I haven't written much about it on my blog yet, but I'm fully converted away from git to jj. I'll say more about this in the future, but one major difference between the two is that jj has anonymous branches. If, like me, you are a huge fan of git, this sounds like a contradiction. After all, the whole thing about branches are that they're a name for some point in the DAG. How do you have a nameless name?

Here's some output from `jj log`:

Here's some sample output from jj log:

```
$ jj log --limit 5
@  pzoqtwuv steve@steveklabnik.com 2024-03-01 15:06:59.000 -06:00 9353442
│  added some cool new feature
│ ◉  xrslwzvq steve@steveklabnik.com 2024-02-29 23:06:23.000 -06:00 a70d4
├─╯  create hello and goodbye functions
│ ◉  yykpmnuq steve@steveklabnik.com 2024-02-29 23:03:22.000 -06:00 21028
├─╯  add better documentation
◉  ootnlvpt steve@steveklabnik.com 2024-02-28 23:26:44.000 -06:00 b5db794
│  only print hello world
◉  nmptruqn steve@steveklabnik.com 2024-02-28 23:09:11.000 -06:00 90a2e97
│  refactor printing
```

Here, we are working on change `pzoqtwuv`. (`@` means the working copy.) There are colors in the real CLI to make the differences more obvious, and to show you unique prefixes, so for example, you probably only need `p` or `pz` instead of `pzoqtwuv` to uniquely identify the change. I'll use the full IDs since there's no syntax highlighting here.

We have two anonymous branches here. They have the change IDs of `xrslwzvq` and `yykpmnuq`. The log output shows the summary line of their messages, so we can see "create hello and goodbye functions" on one branch, and "add better documentation" on the other.

You don't need an additional branch name: the change ID is already there. If you want to add even more better documentation, `jj new yykpmnuq` (or again, likely `jj new yy` in practice) and you're off to the races. (jj new makes a new change off of the parent you specify.)

(And if you're in a larger repo with more outstanding branches, you can ask `jj log` to show specific subsets of commits. It has a powerful DSL that lets you do

so. For example, say you only want to see your commits, `` `jj log -r 'mine()'` `` can do that for you.)

That's all there is to it. We already have the commit messages and IDs, giving an additional identifier doesn't help that much. In practice, I haven't missed named branches at all. And in fact, I kind of really appreciate not bothering to come up with a name, and then eventually remembering to delete that name once the PR lands, stuff like that. Life is easier.

Utility CSS

Another technology I have learned recently is tailwind. But Tailwind is just one way of doing a technique that has a few names, I'm going to go with "utility CSS". The idea is in opposition to "semantic CSS." To crib an example from a blog post by the author of Tailwind (which does a better job of thoroughly explaining why doing utility CSS is a good thing, you should go read it), semantic CSS is when you do this:

```
<style>
  .greeting {
    text-align: center;
  }
</style>

<p class="greeting">Hello there!</p>
```

Whereas, with Tailwind, you end up instead having something like this:

```
<p class="text-center">Hello there!</p>
```

We don't have a new semantic name `` `greeting` ``, but instead describe what we want to be done to our element via a utility class.

So the thing is, as a previous semantic CSS enjoyer, this feels like using inline styling. But there's a few significant differences. The first one is, you're not writing plain CSS, you are re-using building blocks that are defined for you. The abstraction is in building those utility classes. This means you're not writing new

CSS when you need to add functionality, which to me is a great sign that the abstraction is working.

It's also that there is some sleight of hand going on here, as we do, on another level. An objection that gets raised to doing things this way is "what happens when you need to update a bunch of similar styles?" And the answer for that is components. That is, it's not so much that utility CSS says that semantic names are bad, it's that semantic names *at the tag level* are the wrong level of abstraction to use names. To sort of mix metaphors, consider the lambda/closure example. Here's a random function in Rust:

```
fn calculate(list: &[i32]) -> i32 {
    list.into_iter()
        .filter(|i| **i % 2 == 0)
        .sum()
}
```

The `**` is unfortunate, but this function takes a list of numbers, selects for the even ones, and then sums them. Here, we have a closure, the argument to `filter`, but it's inside a named function, `calculate`. This is what using Tailwind feels like to me, we use names for higher level concepts (components), and then keep things semantically anonymous for some of the tasks inside of them (markup).

Heck, even the most pro-semantic-styles folks don't advocate that you must give every single element a class. Everyone recognizes the value of anonymous things sometimes, it's just a matter of what level of abstraction deserves to get named.