

# Updating the whole world to a new API

---

otherwise known as *"eh. I'll get to it"*



# New talk, who dis?

---



- Ben Ford
- Developer Advocate @ Puppet
- @binford2k: [ (<https://www.twitter.com/binford2k>) [ (<https://www.github.com/binford2k>) [ (<https://puppetcommunity.slack.com/team/U11HA7VJ7>) ] ] ]

- 
- Hi, I'm Ben, aka binford2k on the interwebs.
  - I push the bits and talk to the people at Puppet and in our community.
  - If you've been around long enough, then I've probably told you to *just fucking upgrade already* at some point, so in keeping with tradition, I'll be doing that again today.
  - But this time I'm talking about ways that we are working to make it easier for you.

# Anyone recognize this counter?

---



0

Years

0

Months

0

Days

0

Hours

0

Minutes

0

Seconds

Enable Guido Mode

- 
- Does anyone recognize this counter?
  - It's the python 2.7 end of life counter.
  - And as you can see, it's expired.

# Python 2.7 has finally been retired

---

## Long overdue

```
#!/usr/bin/env python  
print ('Hello World!')
```

- January 1994: Python 1.0 meets the world
- October 2000: 2.0 release
- December 2008: 3.0 release

## 2015: Python 2.0 sunset

---

- That's a SEVEN YEAR sunset plan.

# Python 2.7 has finally been retired

---

## Long overdue

```
#!/usr/bin/env python  
  
print ('Hello World!')
```

- January 1994: Python 1.0 meets the world
- October 2000: 2.0 release
- December 2008: 3.0 release



We did not want to hurt the people using Python 2. So, in 2008, we announced that we would sunset Python 2 in 2015, and asked people to upgrade before then. Some did, but many did not. So, in 2014, we extended that sunset till 2020.

## Postponed sunset date of *January 1, 2020*

---

- From the release of 3.0 to the ultimate retiring of the legacy 2.0 line is TWELVE YEARS.

# Python Software Foundation: Press Release 20-Dec-2019

FOR IMMEDIATE RELEASE: 2019-12-20

## **PYTHON 2 SERIES TO BE RETIRED BY APRIL 2020**

CPython core development community urging users to migrate to Python 3 as it will be the only version that will be updated for bugs and security vulnerabilities.

The CPython core developer community is retiring the Python 2 series after nearly 20 years of development. The last major version 2.7 will be released in April 2020, and then all development will cease for Python 2. Users are urged to migrate to Python 3 to benefit from its many improvements, as well as to avoid potential security vulnerabilities in Python 2.x after April 2020. This move will free limited resources

- 
- Even with a 12 year offramp, the end still came abruptly.
  - Four month reprieve. Again.
  - This brand new Mac I'm running, with the latest operating system still defaults to python 2.7
  - So does Debian stable.
  - So does CentOS
  - ...

# Let's update that clock...

---



0

Years

2

Months

25

Days

11

Hours

3

Minutes

46

Seconds

Enable Guido Mode

# This talk is not..

---

## Some things that I'm not talking about today



So far, this sounds like I'm dishing on Python. I want to be clear that I don't intend it this way at all. They're amazing and a victim of their own success. This is a good problem to have.

- Putting down legacy software.
- Downplaying the operational cost of upgrading.
- Difficulty of maintaining backwards compatibility.
- Move fast and break things.

# This talk is not..

---

## Some things that I'm not talking about today



So far, this sounds like I'm dishing on Python. I want to be clear that I don't intend it this way at all. They're amazing and a victim of their own success. This is a good problem to have.

- Putting down legacy software.
- Downplaying the operational cost of upgrading.
- Difficulty of maintaining backwards compatibility.
- Move fast and ~~break things~~.
  - Updated: ^ DON'T break things.

# Maintenance is really hard

---

## For projects and for infrastructures both

- Cost of supporting old features grows exponentially over time.
- Reduces the pace for improvements.
- Means that new features are harder to use.
  - Convolutd to continue supporting old functionality too.
- Breaking changes break infrastructure.
  - Must expend effort just to get back to where you were.



Example:

it's hard to make collectors work better when people depend on their two very different and unrelated side effects!

```
# Ensure our internal mirror is configured before we install any packages
# ... but their side effect also realizes all virtual packages!
Yumrepo <| |> -> Package<| |>
```

# Maintenance is really hard

---



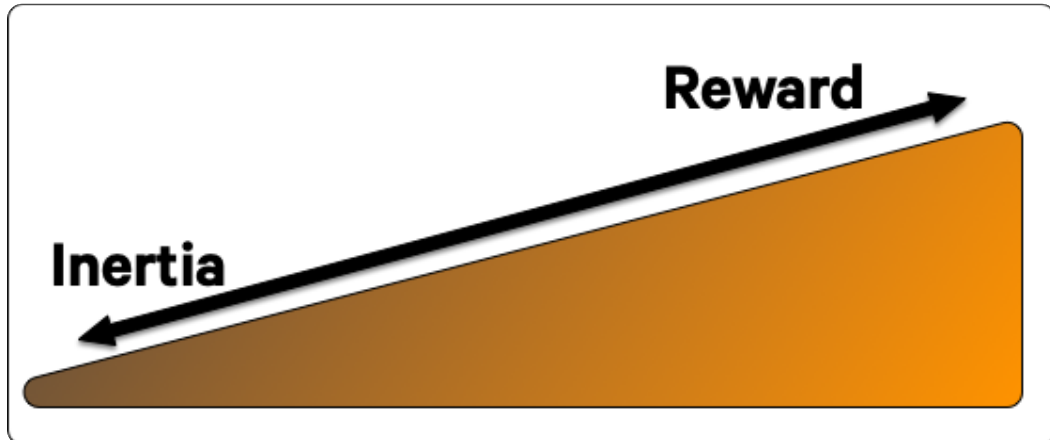
Sometimes it feels like we're paddling really hard just to stay where we are.



# Motivation for change

---

Or not.



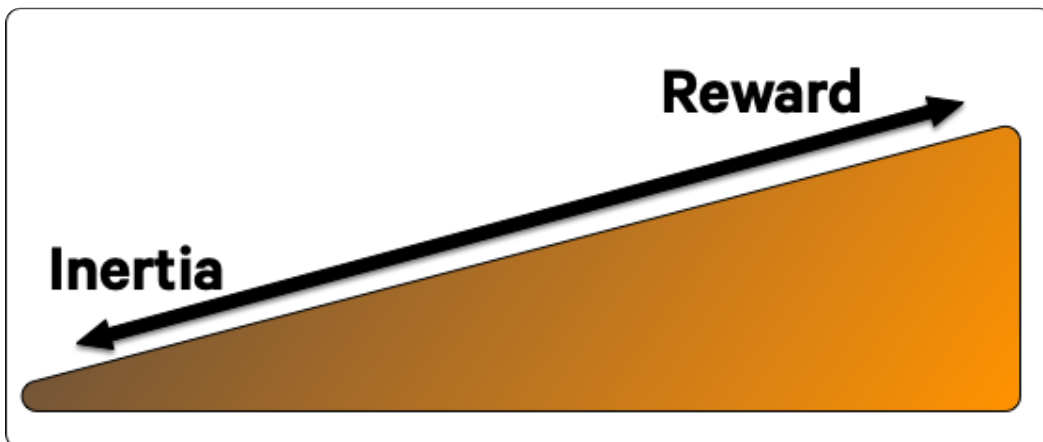
- Inertia
  - Resistance to change is a real thing
  - Why upgrade when it works fine?
- Reward
  - New features.
  - Works better/faster/safer.

- 
- Often upgrades offer nothing but the chance to make mistakes and break things!

# Motivation for change

---

Or not.



- Inertia
  - Effectively a CONSTANT for any given community.
  - Little control over this value.
- Reward
  - The amount of reward we can offer in a new release is finite.
  - Users want new exciting features, but without changing anything.
  - Ultimately, little control over this value either.

- 
- Inertia:
    - Early adopters have very low inertia and are happy to chase the bleeding edge.
    - Late adopters or enterprise customers value stability and resist change.
    - The only way we can control *inertia* is by targeting a different community.
    - Education doesn't change what the institution values.
  - Reward:
    - The cost of adding new features grows exponentially as a project matures.
    - Often we pay for it in stability, which ends up as a detractor again.
    - And there are limits to what we can build in any given release cycle.

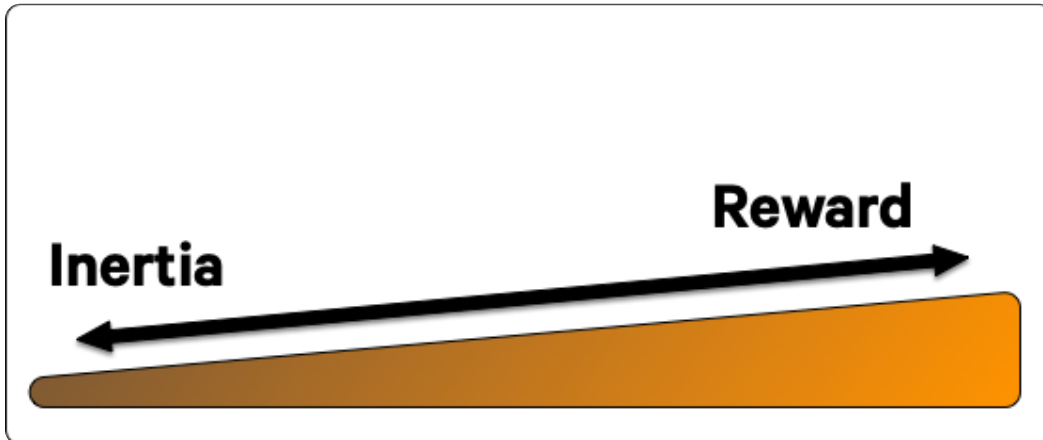


Ultimately, this means that the differential between inertia and reward for a minor upgrade cycle is effectively fixed and pretty small.

# Friction

---

But there's a third factor



Friction is the factor that makes forward motion more difficult. I'm representing it as the slope in this drawing. And this we *can affect*.

- 
- Make the upgrade path easier, more predictable, and more reliable.
  - Build trust in the transitions.
    - Admittedly, in the earlier days Puppet wasn't great at this.
  - Automate what we can, so the mental cost is lower.
    - Example: database backed apps generally handle schema updates for you.

# Puppet 3.x functions

---

An example of this in action.

```
module Puppet::Parser::Functions
  newfunction(:strlen, :type => :rvalue) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

- Designed for a simpler time.
- Environment safety wasn't yet a concern.
- Didn't understand the cost of polluting the global namespace.
  - Module ecosystem wasn't as rich, so collisions were rare.
- Manhandling arguments marshalled into an array didn't seem like a big deal.
- Weird `:rvalue` parameters were fine; bleeding edge, remember?
  - Later on we added `:arity` too. What does that even mean?

- 
- `rvalue` and `arity` are well known terms to computer scientists.
    - Most of us are not computer scientists.
  - Polluting the global namespace meant that any function could easily stomp on any other -- or even interfere with Puppet internals.
  - Leaking across environments meant that functions often leaked into production long before we intended them too.

# Puppet 3.x functions

---

An example of this in action.

```
module Puppet::Parser::Functions
  newfunction(:strlen, :type => :rvalue) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

- But they were easy to write.
- Cut & pasted boilerplate.
- We got used to the warts.

# Puppet 3.x functions

---

An example of this in action.

```
module Puppet::Parser::Functions
  newfunction(:strlen, :type => :rvalue) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```



---

Eh. They're good enough.

# Modern Function API

---

Elegantly solves all those problems.

- Environment safe!
- Thread safe!
- No global namespace pollution!
- Data typed function signatures!



# Easy to port

---

First identify components:



Function name.

```
module Puppet::Parser::Functions
  newfunction(:strlen,
    :type => :rvalue,
    :doc => "Just a naive strlen example",
  ) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

# Easy to port

---

First identify components:



Documentation.

```
module Puppet::Parser::Functions
  newfunction(:strlen,
    :type => :rvalue,
    :doc => "Just a naive strlen example",
  ) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

# Easy to port

---

First identify components:



Parameter validation and handling.

```
module Puppet::Parser::Functions
  newfunction(:strlen,
    :type => :rvalue,
    :doc => "Just a naive strlen example",
  ) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

# Easy to port

---

First identify components:



Implementation.

```
module Puppet::Parser::Functions
  newfunction(:strlen,
    :type => :rvalue,
    :doc => "Just a naive strlen example",
  ) do |args|
    raise "Wrong number of args" unless args.size == 1
    raise "Wrong type of args" unless args.first.is_a String

    args.first.length
  end
end
```

# Easy to port

---

Then write replacements:



Function name.

```
# @summary
#   Just a naive strlen example
Puppet::Functions.create_function(:'mymod::strlen') do
  # @param value
  #   The string to calculate the length of
  # @return [Integer]
  #   The length of the input string
  dispatch :default_impl do # invoke default_impl method when matched
    param 'String', :value
  end

  def default_impl(value)
    value.length
  end
end
```

# Easy to port

---

Then write replacements:



Documentation.

```
# @summary
#   Just a naive strlen example
Puppet::Functions.create_function(:'mymod::strlen') do
  # @param value
  #   The string to calculate the length of
  # @return [Integer]
  #   The length of the input string
  dispatch :default_impl do # invoke default_impl method when matched
    param 'String', :value
  end

  def default_impl(value)
    value.length
  end
end
```

# Easy to port

---

Then write replacements:



Parameter validation and handling.

```
# @summary
#   Just a naive strlen example
Puppet::Functions.create_function(:'mymod::strlen') do
  # @param value
  #   The string to calculate the length of
  # @return [Integer]
  #   The length of the input string
  dispatch :default_impl do # invoke default_impl method when matched
    param 'String', :value
  end

  def default_impl(value)
    value.length
  end
end
```

# Easy to port

---

Then write replacements:



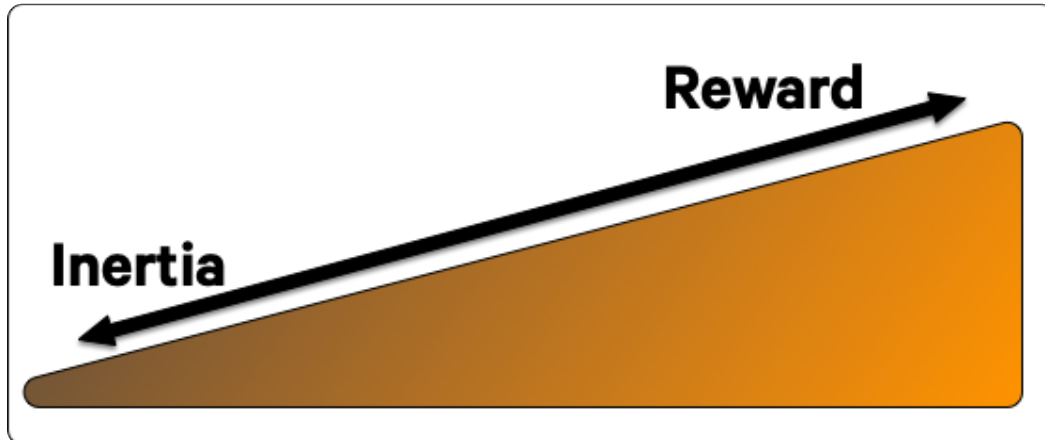
Implementation.

```
# @summary
#   Just a naive strlen example
Puppet::Functions.create_function(:'mymod::strlen') do
  # @param value
  #   The string to calculate the length of
  # @return [Integer]
  #   The length of the input string
  dispatch :default_impl do # invoke default_impl method when matched
    param 'String', :value
  end

  def default_impl(value)
    value.length
  end
end
```

# Let's come back to inertia for a bit

## Drivers or blockers to change



- We showed the rewards already.
- Elimination of some major pain points.
- But the module author doesn't necessarily have those pain points.
- And the end user doesn't always see the benefits.



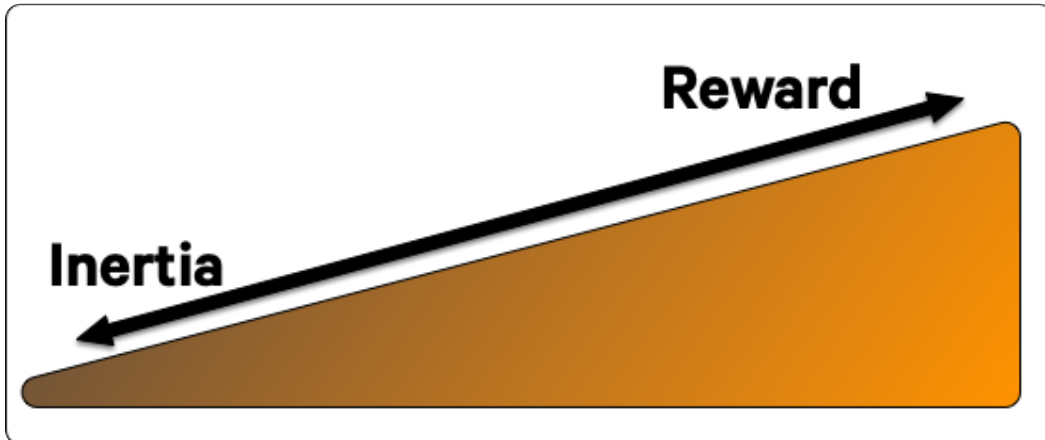
Inertia is constant and significant. And rewards are actually quite small.

- 
- Any change is more work than doing nothing at all.
    - We don't update just to upgrade.
    - If it already works, upgrading without improvements is just a opportunity to break something.
  - The differential between the cost of upgrading and the benefits of upgrading is vanishingly small.
  - This means that we have a very small window to provide incentives to upgrade.
    - Which is why people are still using Python 2.7.

# What about upping the reward?

---

How much more motivation can we provide?



- Surface pain points via warnings.
- Add functionality that doesn't exist in earlier API.
- Provide better documentation via `puppet-strings`.
- ... not really a lot ....



When inertia and friction is greater than reward, we have stasis.

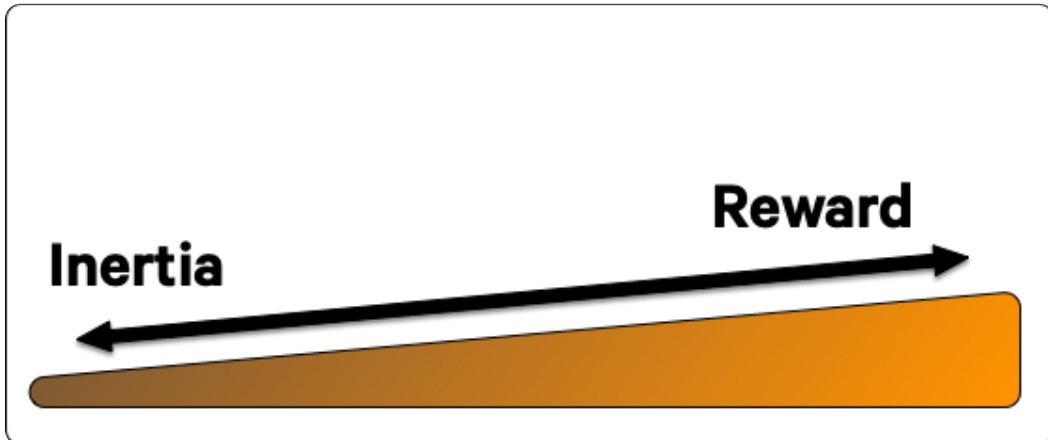


This means that new functions are often written to the new API, but very few people bother to upgrade their existing functions!

# Friction

---

So what can we do to lower friction?



- Make the upgrade path easier, more predictable, and more reliable.
- Show how to do it with education.
  - Publishing tutorials and blog posts.
- Automate what we can, so the mental cost is lower.



Can I programatically identify the components we just looked at?

---

Since we've only got a tiny window to work with, let's not waste it with friction.

# Yes we can!

---

(well mostly)



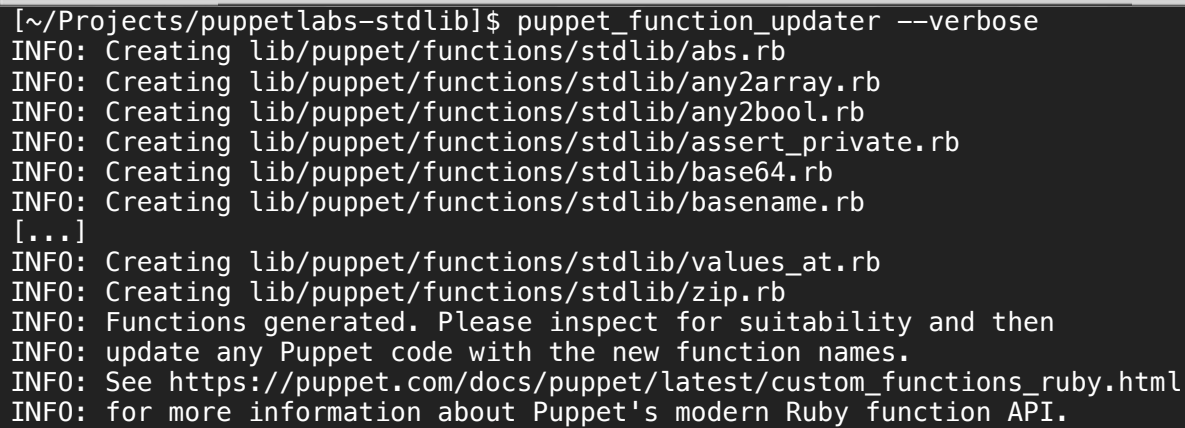
It turns out that identifying the parts of a Puppet 3.x function that we care about is almost completely automatable.

- Over the American Thanksgiving holiday, I had a little free time.
- This was a problem that had intrigued me so I dove in.
- Ruby makes it surprisingly easy to introspect into source code.
- As is tradition, I built a cleverly named tool to assist in the update.
  - <https://github.com/binford2k/puppet-function-updater>
  - <https://binford2k.com/2019/11/27/automagic-function-port/>

# Parses function source files

---

## Generates replacement modern API function



```
[~/Projects/puppetlabs-stdlib]$ puppet_function_updater --verbose
INFO: Creating lib/puppet/functions/stdlib/abs.rb
INFO: Creating lib/puppet/functions/stdlib/any2array.rb
INFO: Creating lib/puppet/functions/stdlib/any2bool.rb
INFO: Creating lib/puppet/functions/stdlib/assert_private.rb
INFO: Creating lib/puppet/functions/stdlib/base64.rb
INFO: Creating lib/puppet/functions/stdlib/basename.rb
[...]
INFO: Creating lib/puppet/functions/stdlib/values_at.rb
INFO: Creating lib/puppet/functions/stdlib/zip.rb
INFO: Functions generated. Please inspect for suitability and then
INFO: update any Puppet code with the new function names.
INFO: See https://puppet.com/docs/puppet/latest/custom\_functions\_ruby.html
INFO: for more information about Puppet's modern Ruby function API.
```

- Ports old patterns to new patterns
- Generates a basic spec test for each function
- Validates each function and warns on invalid output

- 
- I'm sure that some of you will be disappointed that I'm not going into tech details about how this thing works.
  - But my slot doesn't give me time for it.
  - I'll be happy to talk later, or tomorrow at our Contributor session.

# Ports *most* of the function cleanly

---

## Legacy API doesn't capture the function signature



Remember that the legacy API just passed all arguments as a single untyped array and relied on the programmer to know what to do with that.

- The only part of the function that I cannot infer reliably.
- So instead, I create a `dispatch` that does the same thing.

```
dispatch :default_impl do
  # Call the method named 'default_impl' when this is matched
  # Port this to match individual params for better type safety
  repeated_param 'Any', :args
end
```

```
def default_impl(*args)
  # parameter handling and implementation copied in
end
```

# Then I got a little crazy

---

What if we could update the whole world?



# Then I got a little crazy

---

What if we could update the whole world?



GitHub exposes a fantastic BigQuery dataset of all their public repositories.

Generate a list of all Puppet modules with legacy functions:

```
SELECT DISTINCT repo_name
FROM `bigquery-public-data:github_repos.files`
WHERE STARTS_WITH(path, 'lib/puppet/parser/functions')
AND ref = 'refs/heads/master'
```

# Validate the tool

---

## First let's build some confidence

1. First I wrote a quick script to clone each listed repo.
2. Then it ran my tool to port functions.
3. If successful, it deleted the repo.

I was left with 47 modules with edge cases to account for, but after a couple iterations and improvements, it ported all valid functions flawlessly.



---

So then I updated the script with a couple more steps, then ran it and went out for the night.

1. Instead of just cloning, now it forked too.
2. It used the GitHub API to ignore forks and only fix upstream modules.
3. After porting the function, it would submit a PR with instructions on how to complete the job.
4. And then cleaned up by deleting the repo from my namespace.

# Next steps

---

## Finishing the port



The automated pull request included links to the [accompanying tutorial](#) that describes these steps on how to finish the port.

## Left as an exercise for the author:

- Port manifests to use the namespaced function name.
  - Or change it back to top-level (only if you're `stdlib!`)
- Add one or more dispatches with proper data types.
- Convert (or write) documentation to `puppet-strings` format.
- Write more tests following the example provided.

- 
- The port is effectively bug-for-bug.
  - Can't magically infer the parameter handling.
  - Just copied docs straight over.
  - Validated code, but doesn't take advantage of new features.
  - Note that changing the function name makes it a ***semver breaking change*** even if functionality is exactly the same.

<https://binford2k.com/2019/11/27/automagic-function-port/>

# Mixed Response

---

## But mostly positive

Some were sort of neutral:



Cute idea, however I'm not going to merge it unless someone gives it a smoke test at least.

Some were very positive:



I love this initiative by @binford2k ❤️ #puppetize #opensource

# Mixed Response

---

## But mostly positive

But the two biggest annoyances were:

1. My fork detection didn't work properly and it contributed PRs to forks instead of just upstream projects.
2. It created detached pull requests, which meant that module owners couldn't just add to the PR to get a self-contained function port. They had to *merge it* and then add more commits to flesh it out.



---

Also, nobody asked or suggested it, but I should have DCO signed the commits.

# Community Engagement

---

## Developer to developer

- Authenticity is key.
- Reaching out to your users is *always* valued.
  - Maybe it actually helped improve someone's code.
  - Or it just provided a reminder to do it.
  - Or even didn't help directly, but showed someone *how* to do it.
- Even in the worst case, it's a signal that we care.
- This connection keeps us all invested in each other and our community.



I'm grateful to all of you and to the culture we have built here. Even when a contribution isn't perfect, it's recognized and valued as a genuine desire to help improve things. I credit **Vox Pupuli** for a lot of that.

- 
- I honestly expected some pushback from people annoyed by "garbage" PRs
  - Turns out that authentic connections with a developer community leads to an assumption that things are coming from the right place.
  - And people appreciate genuine outreach as long as it's driven by shared value.
    - Sure, it benefits Puppet when you all upgrade, but it benefits you too, and that's the real reason I'm doing this!

# Getting better over time

---

## We've got a small history of these things now

- Manually fixed 5–10 modules broken by a `puppetlabs/mysql` update.
- Discovered a crashing bug when a module had a zero-byte `init.pp` file.
  - Identified all the 218 modules affected.
  - Ranked and evaluated by usage and ecosystem impact.
  - Scripted pull request run instead of yanking the release.
  - Fun fact: I ran this from the floor of *Puppetize Live: Amsterdam*.
- We're doing this 2–3 times a year now.
- Learning each time, maybe we can do more?

# What else?

---

## Future developer enablement

- What other ways could we engage and enable?
- Do we like the idea of offering PR fixes?
- Build a framework for more one-off runs like this.
  - Automatically upgrade Hiera 3 hierarchies to Hiera 5?
  - Port known facts to the modern `fact()` function?
- Other parts of Puppet are following suit:
  - The Bolt project automatically updates your inventory file.



Notice the giant ribbon in the corner? These handful of slides are just me spitballing the future! Don't expect these to be promises.

SPECULATION!

# Puppet Developer Kit

---

## Create PDK plugins for update scripts

- A full plugin system for the PDK is near.
- Perhaps the update scripts should go there.
- The PR framework can invoke the PDK.
- Or module authors can use it themselves.

SPECULATION!

# Forge tie ins?

---

The Forge is rad. Can we make it moreso?

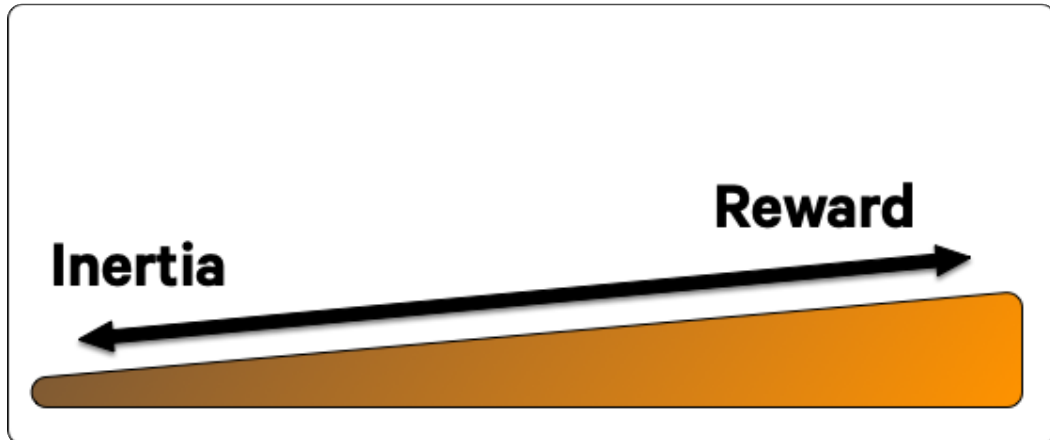
- Maybe opt-in on Forge profile page.
- Instead of scraping GitHub, let authors choose.
- Alerts for out-of-date platform or Puppet dependencies.
  - Shoot, maybe we could even validate for you.
  - Generate tested platform support instead of requiring you declare it.
- Warnings when modules you depend on are updated.

SPECULATION!

# Which brings us back to friction

---

and how we can lower it.



My job is to lower your friction as a developer.

- Your main job isn't to write Puppet code.
- Your job is to architect, build, deploy, maintain, update infrastructure.
- Puppet is just one tool that helps you do that job.
  - And my job is to make that easier for you.

# Provide a path

---

And give you the tools to do so.

Other irons in the fire now.

- Rangefinder code change impact analysis
  - What downstream usage could a pull request affect?
- Content usage telemetry project
  - How many people are using which classes and types from your modules and on what platforms?
  - For this, I'll need your help socializing so people adopt it.



Most importantly, both of these tools are powered by *open datasets* that are finally just weeks from going public. You'll be able to use the same data to invent other tooling.

- 
- <https://github.com/puppetlabs/puppet-community-rangefinder>
    - Example: <https://github.com/puppetlabs/puppetlabs-postgresql/pull/1132>
  - Telemetry implementation is not published yet
    - Hacking on it during Contributor event tomorrow
    - Want feedback on data collection and privacy

# And ultimately

---

We are working to make your job as contributors easier



Our goal is that maintaining quality should be easier than not.



- 
- You'll notice that my handle is on every one of these slides.
  - That's because I want to hear from you.
    - feedback, suggestions, ideas.

# A quick summary

---

## Motivation for change and initiatives

- Velocity of change is driven by three things.
  - *Inertia* holds you back.
  - Promise of *Rewards* pulls you forward.
  - And *Friction* slows or prevents change.
- Project maintainers have little control over inertia or rewards.
- But we can work to decrease friction.



These points apply to you in your own open source projects too.

---

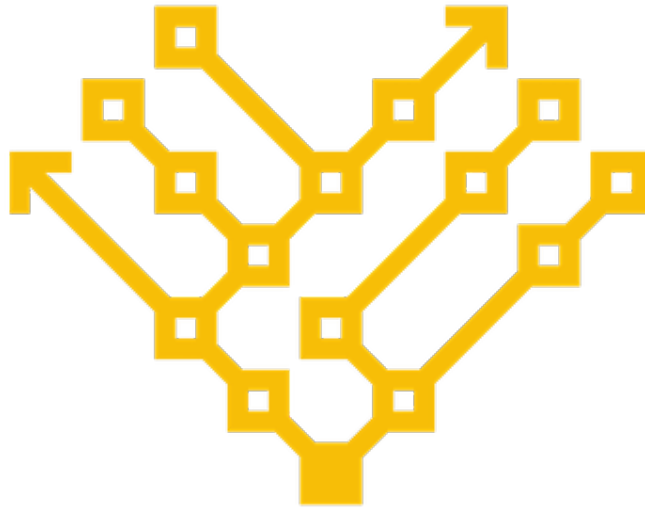
As both a consumer and as a producer.

# A quick summary

---

## Motivation for change and initiatives

- Some ways that Puppet will be reducing friction for you:
  - Suggesting fixes/updates for known issues.
  - Providing tooling that helps make updates.
  - Forge integrations to surface things that can be improved;
  - and ways that we can help each other with those improvements.



# Learn More

---

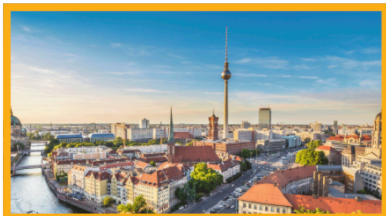
## The info dump page

- [Porting functions](#) to the new API:
  - <https://puppet.com/blog/refactoring-legacy-ruby-functions/>
- Check out the automagic [Puppet Function Updater](#):
  - <https://github.com/binford2k/puppet-function-updater>
  - Maybe help me port it to the PDK?
- Blog posts on the updater and next steps:
  - <https://binford2k.com/2019/11/27/automagic-function-port/>
  - <https://binford2k.com/2020/01/11/porting-functions/>
- Read more about [documenting your functions](#) or other Puppet code
  - [https://puppet.com/docs/puppet/latest/puppet\\_strings.html](https://puppet.com/docs/puppet/latest/puppet_strings.html)
- Watch my twitter for a link to this presentation soon.

# Puppet Camps!

---

Share the awesome projects you're working on



## **Berlin**

11 March 2020, 9 am – 5 pm



## **Melbourne**

24 March 2020, 9 am – 5 pm



## **Boston**

8 April 2020, 9 am – 5 pm



CFP for all three is open: [submit a talk!](#)

