Creating a Code Review Culture

Johnathan Turner
@while1malloc0
breaking.computer

Code review is useful

Code review provides

a means of ensuring code quality

Code review provides

a communication platform

Code review provides

an opportunity to teach

A code review *culture* is useful

Culture

"the set of shared **attitudes**, **values**, **goals**, and **practices** that characterizes an institution or organization" - Merriam Webster

Agenda:

Examine the practices that contribute to a strong code review culture from the perspective of...

- Organizations
- Authors
- Reviewers

Organizations

Be intentional about your culture

Be intentional about your culture

by communicating the culture

Be intentional about your culture

by establishing a community of experts

Be intentional about your culture

by developing new experts

Be intentional about your culture

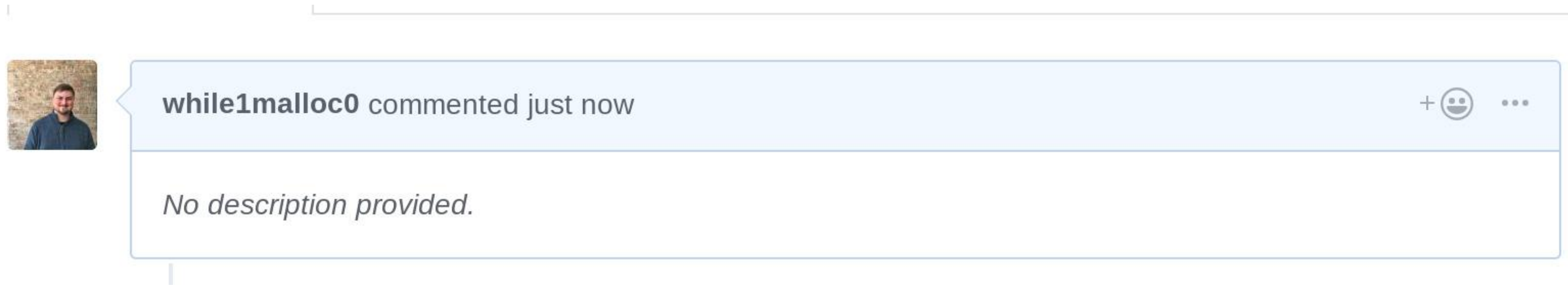
by training code reviewers

Code authors

Make the reviewer's life easier

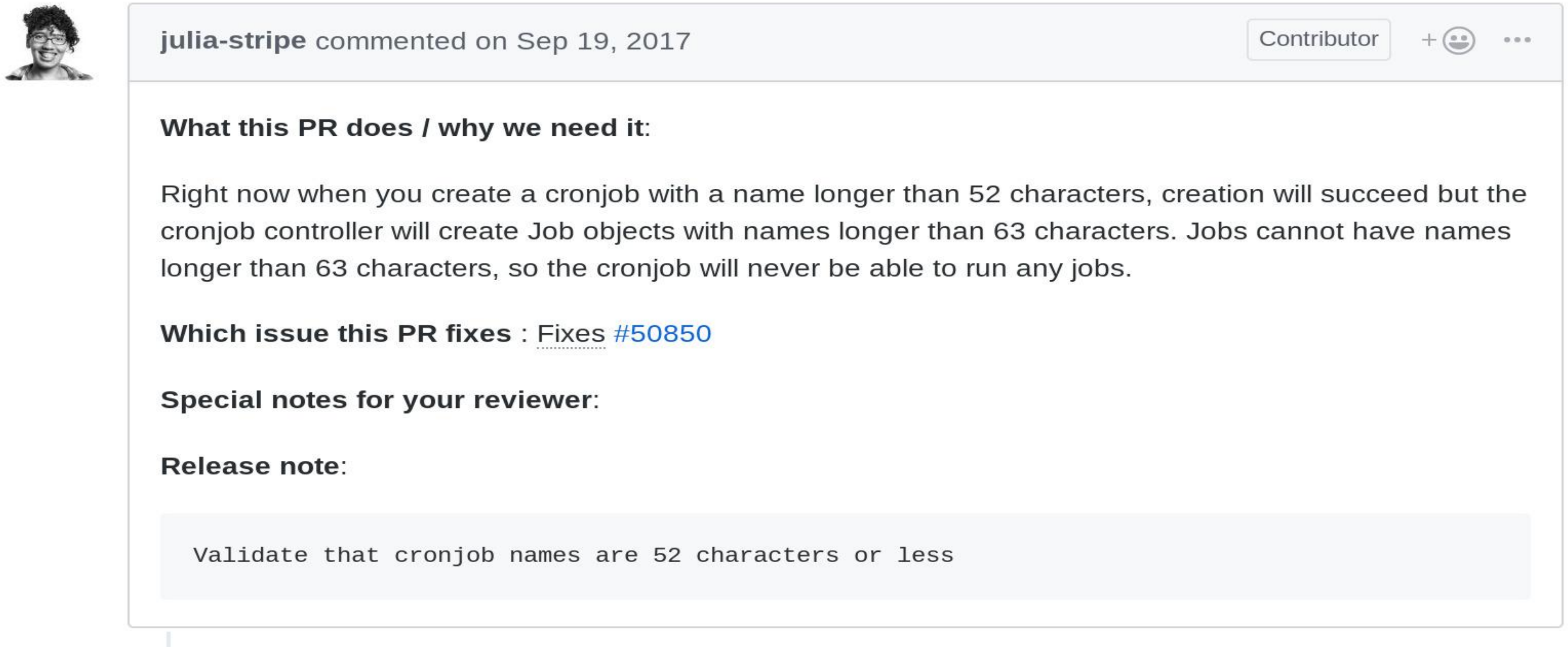
Make the reviewer's life easier

by communicating context



while1malloc0 commented just now

No description provided.


julia-stripe commented on Sep 19, 2017

What this PR does / why we need it:

Right now when you create a cronjob with a name longer than 52 characters, creation will succeed but the cronjob controller will create Job objects with names longer than 63 characters. Jobs cannot have names longer than 63 characters, so the cronjob will never be able to run any jobs.

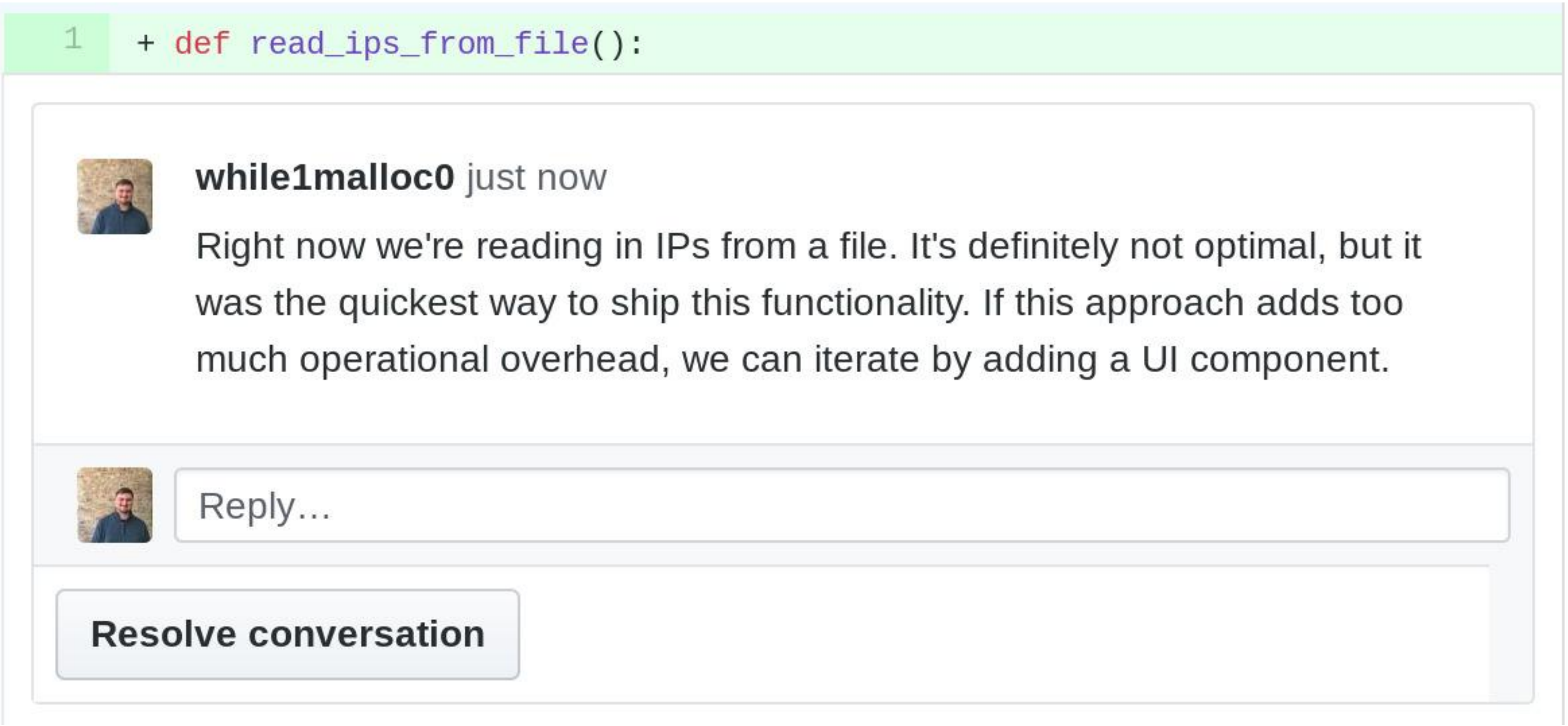
Which issue this PR fixes : Fixes #50850

Special notes for your reviewer:

Release note:

```
Validate that cronjob names are 52 characters or less
```

Photo courtesy of Julia
Evans, @b0rk



```
+ def read_ips_from_file():
```

while1malloc0 just now

Right now we're reading in IPs from a file. It's definitely not optimal, but it was the quickest way to ship this functionality. If this approach adds too much operational overhead, we can iterate by adding a UI component.

Make the reviewer's life easier

by making the PR a manageable size

Vertical Slices

- ship the smallest unit of functionality meaningful to your users
- round trip through your stack

Make the reviewer's life easier

by automating the nits

Make the reviewer's life easier

by knowing when to take it offline

Code reviewers

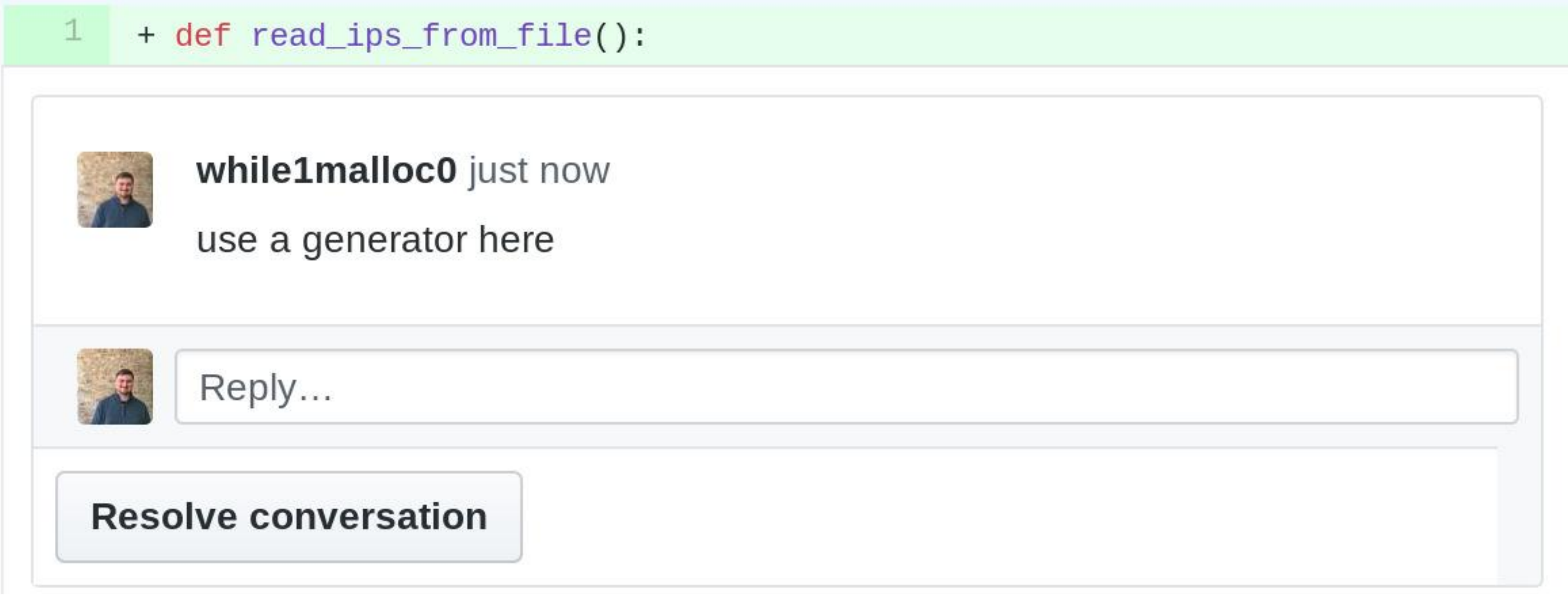
Communicate mutual respect

Communicate mutual respect

by knowing when to take it offline

Communicate mutual respect

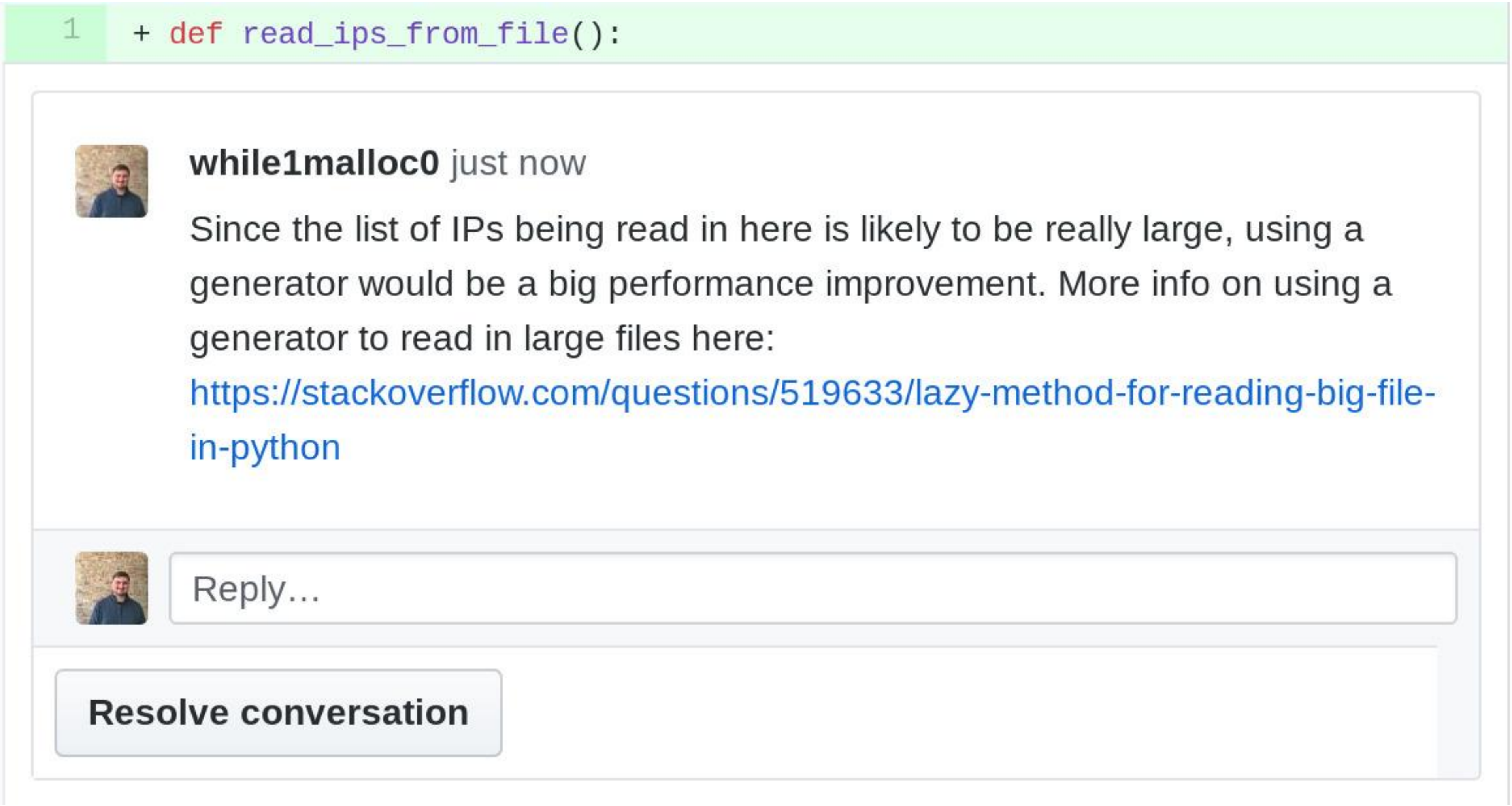
by including justification for critique



```
1 + def read_ips_from_file():
```

while1malloc0 just now

use a generator here



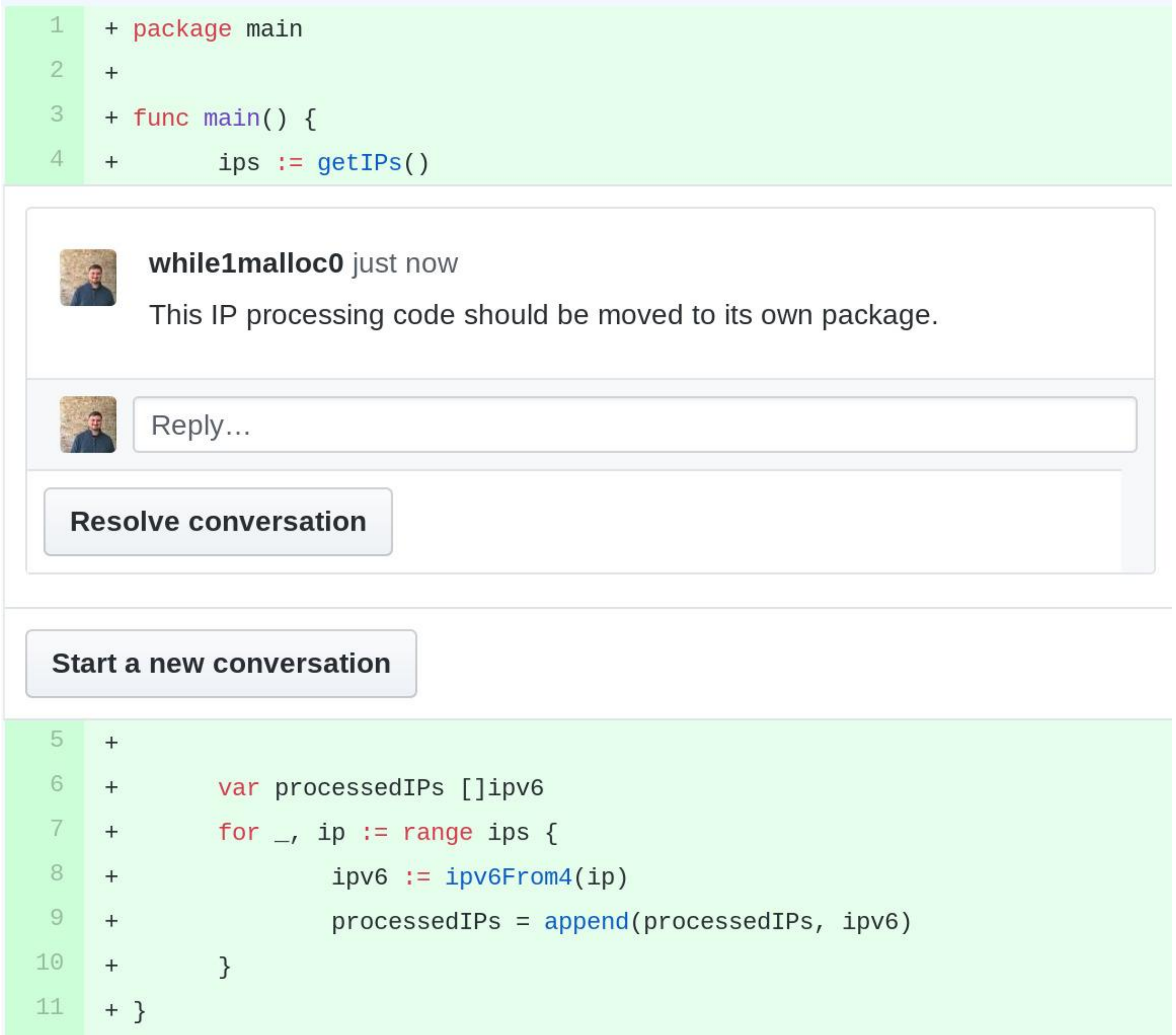
```
+ def read_ips_from_file():
```

while1malloc0 just now

Since the list of IPs being read in here is likely to be really large, using a generator would be a big performance improvement. More info on using a generator to read in large files here: https://stackoverflow.com/questions/519633/lazy-method-for-reading-big-file-in-python

Communicate mutual respect

by engaging with the author as an equal

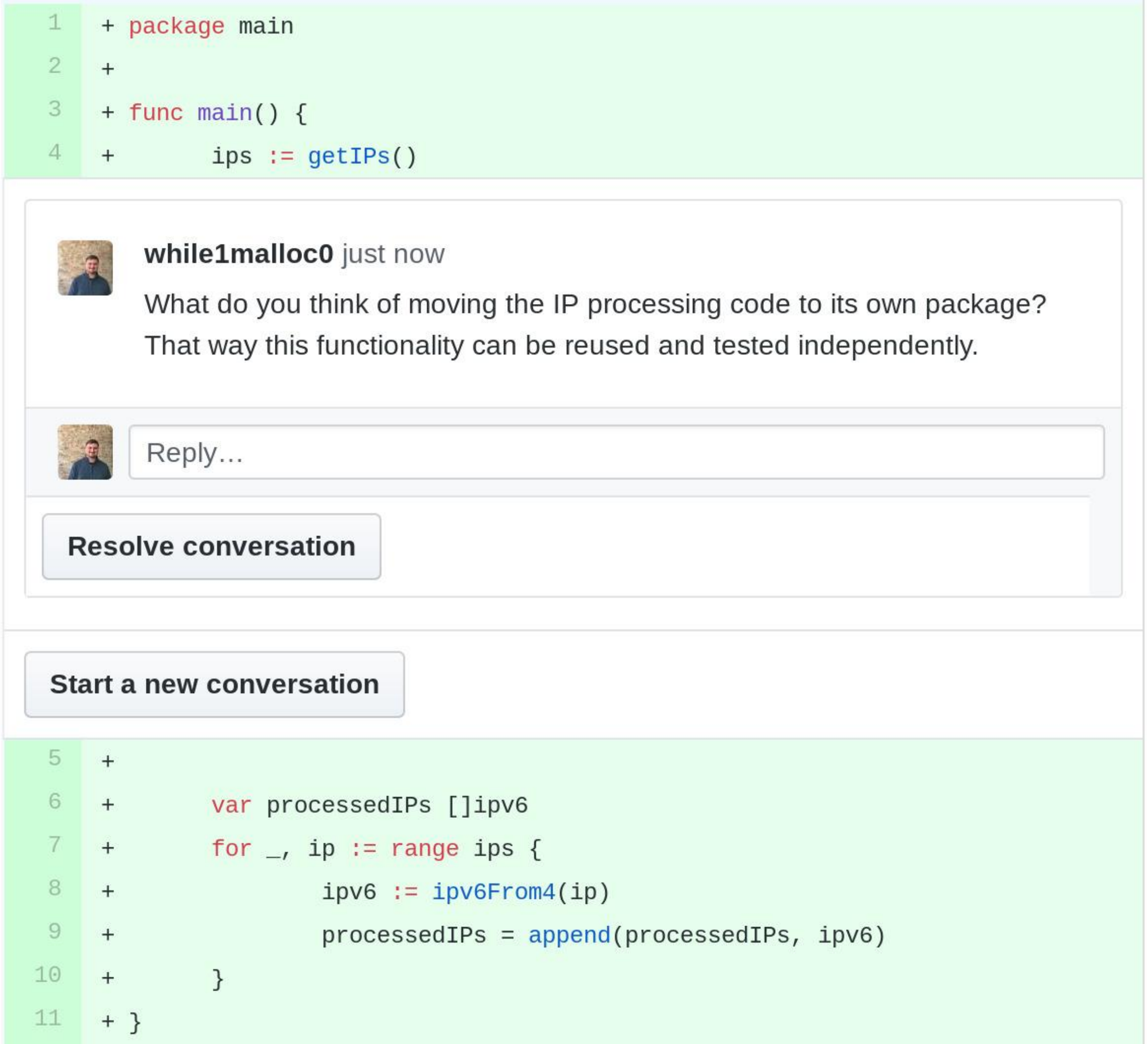


```
+ package main
+
+ func main() {
+       ips := getIPs()
```

while1malloc0 just now

This IP processing code should be moved to its own package.

```
+
+       var processedIPs []ipv6
+       for _, ip := range ips {
+               ipv6 := ipv6From4(ip)
+               processedIPs = append(processedIPs, ipv6)
+       }
+ }
```



```
+ package main
+
+ func main() {
+       ips := getIPs()
```

while1malloc0 just now

What do you think of moving the IP processing code to its own package? That way this functionality can be reused and tested independently.

```
+
+       var processedIPs []ipv6
+       for _, ip := range ips {
+               ipv6 := ipv6From4(ip)
+               processedIPs = append(processedIPs, ipv6)
+       }
+ }
```

Communicate mutual respect

by being as thorough as the PR needs

Reviewing in passes

Each pass is a theme, and some questions to help focus on that theme

Reviewing in passes

Make your own. Make a checklist.

Passes to complete every time

If there are red flags on any of these, resolve before adding more commentary.

Sizing up

- What is the general shape of the PR?
- Is the PR the right size?

Context

- What is this PR trying to accomplish?
- Why is this PR trying to accomplish that?
- Does the PR accomplish what it says?

Relevance

- Is the change made in this PR necessary?
- Does this PR duplicate existing functionality?
- Are there others that should be aware of this PR?

Passes for more in-depth review

Do these for more substantial PRs. Pick the ones relevant to the change.

Readability

- Is the change reasonably understandable by other humans with little/no prior experience of the code?
- Are any esoteric language features being used?

Production readiness

- How will we know when this breaks?
- Is there new documentation required by this change?
- Are there tests that prevent regression?
- Is this change secure?

Naming

- Do names communicate what things do?
- Are the names of things idiomatic to the language?
- Do names leak implementation details?

Gotchas

- What are the ways in which added or changed code can break?
- Is this code subject to any common programming gotchas?
- Is spelling correct and consistent?

Language specific

- Is the code well designed?
- Is the code idiomatic to the language?
- Are new patterns introduced?
- Does the code fall into common pitfalls for the language?

Full checklist at:

github.com/while1malloc0/code-review-checklist

Thank you

Johnathan Turner
@while1malloc0
breaking.computer