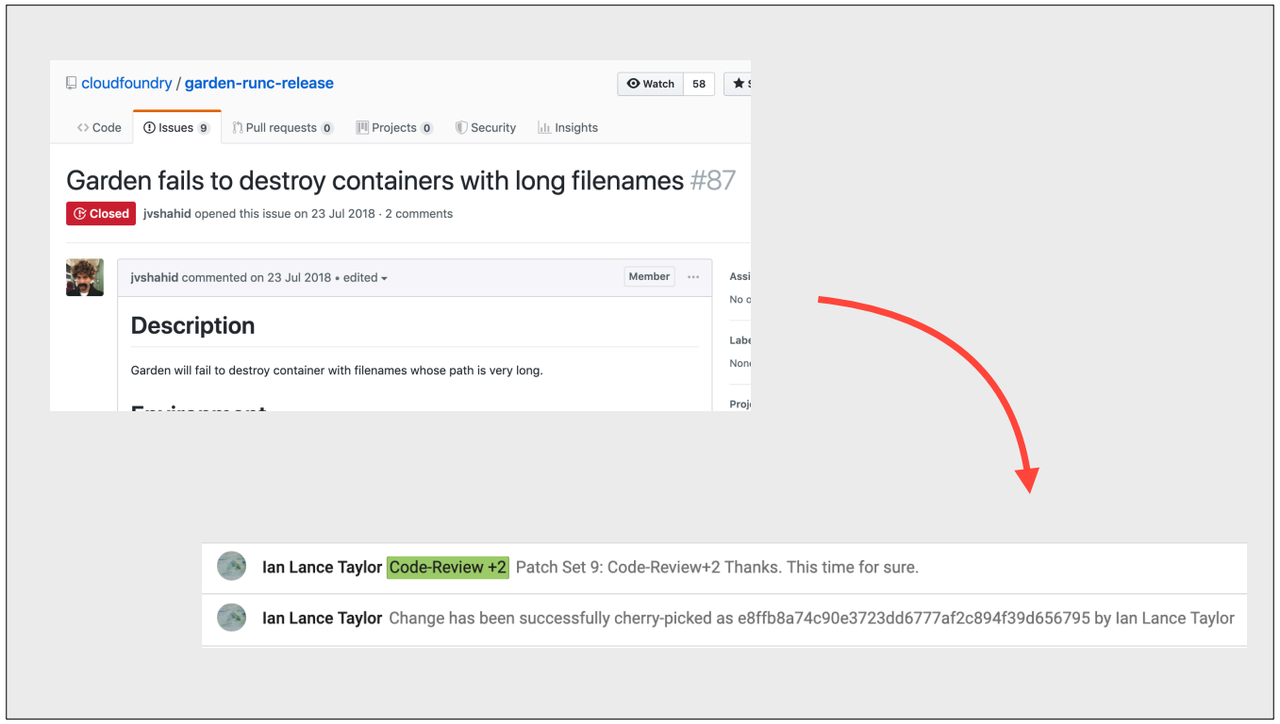
How Deep Do You *Go*?

Contributing to the os Package

Oliver Stenbom

July 25th
Gophercon 2019



On an unsuspecting Monday last July, the team I was working at the time received a bug report.
The report claimed that our delete button wasn't working in certain scenarios.
In our case we were building containers -> which meant that our containers were failing to be destroyed.

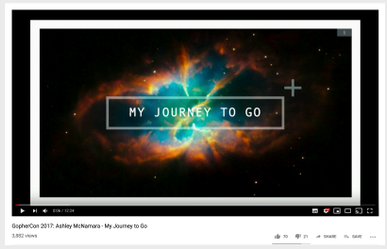
Three months and a tough journey later, this bug report had grown into a contribution to the go language.
This talk is about how that happened, and how *even* novices such as myself can gain from the contribution experience.

Previous Gophercon Contribution Talks

2016

Michael Matloob:
The tools you'll use

2017



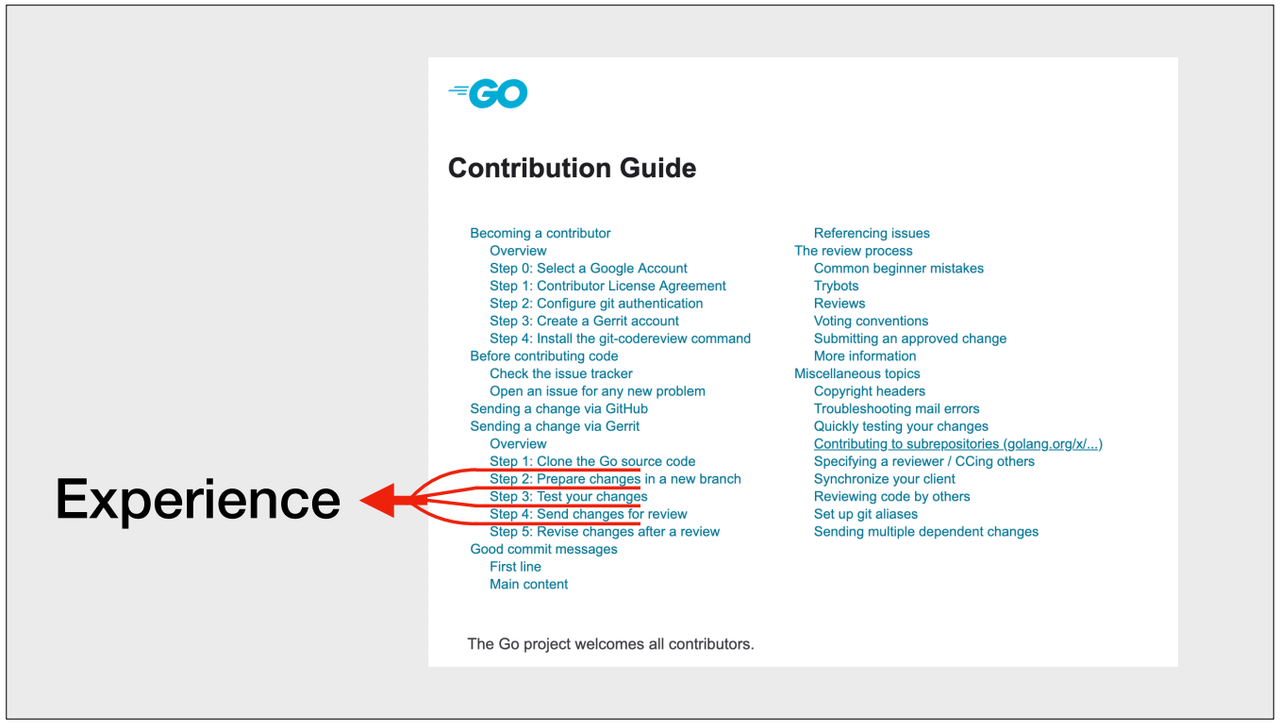
Ashley McNamara:
My Journey to Go

2018

Kevin Burke:
The many ways
to contribute

I am not the first to talk at this conference about contribution. These previous talks can help you along your journey in different ways. They will help you to understand the tools you'll have to use and the different ways it's possible to contribute to go. I can definitely recommend them if you're just getting started.

Hopefully today though. I'm going to address something slightly different.



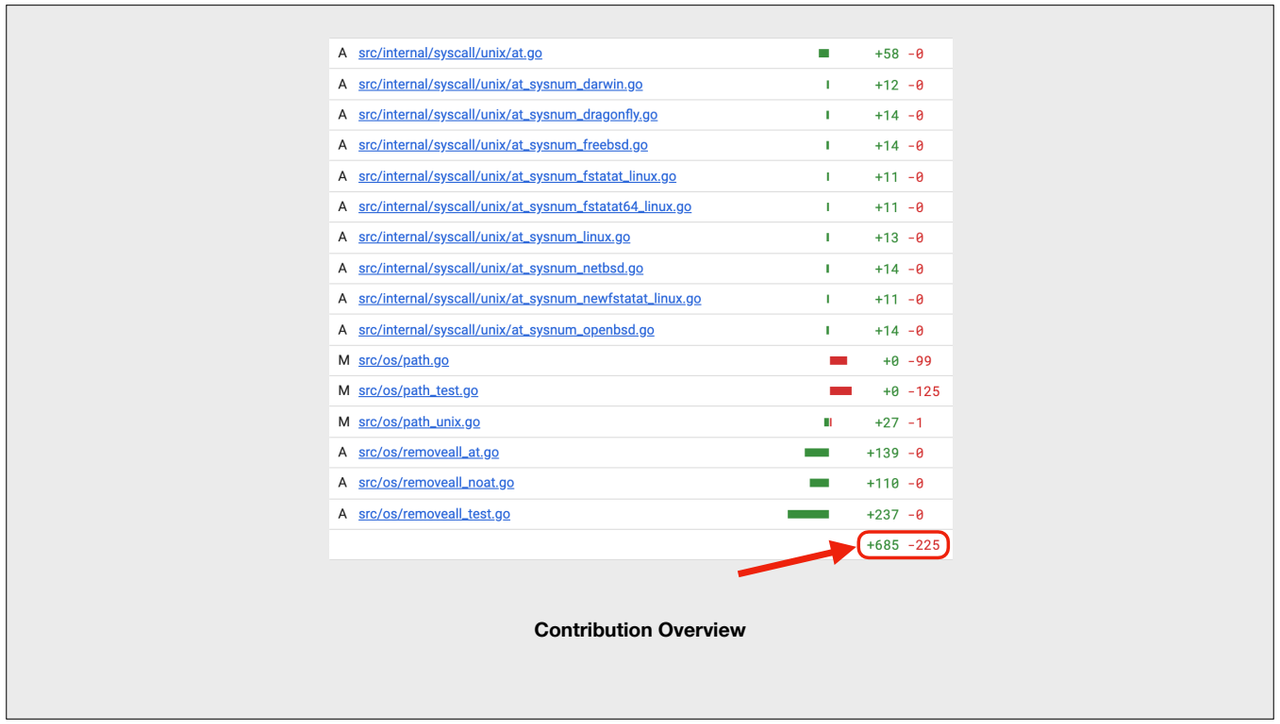
Let’s take a look at the go contribution website. There’s a bunch of info about *how* to contribute. It details all of the *concrete* steps you’ll have to take to add something to go.

But in my eyes there is *a lot* to know that’s hidden *between* the lines. What’s contribution like? How does it come about? What can it offer me and how can I offer something to the language?

We could call this the contribution experience, and I am going to share my version of that experience with you in the next 20 mins.

Disclaimer: Mine is just one of many ways to contribute!

I really want to highlight that this is just one experience of many.



Contribution Overview

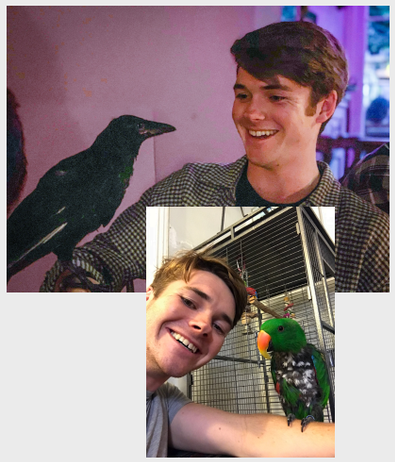
Our contribution ended up being reasonably large, whereas others may be smaller, larger or simply different.

Regardless of that, I hope that my story will help you to understand what you’re in for - to set you up for contribution success.

About Oliver

- Just completed masters @ Imperial College
  - Eliminating serverless cold-starts..
- Last summer on Cloud Foundry @ Pivotal London
  - First line of Go in May

merged contribution in October



My name is Oliver Stenbom
I just completed my masters degree at Imperial College London, where I wrote my thesis about eliminating serverless cold-starts

I spent last summer working on the Cloud Foundry Container team - Garden @ Pivotal London.
There I wrote my first line of Go ever in May and had my first go contribution merged in October.

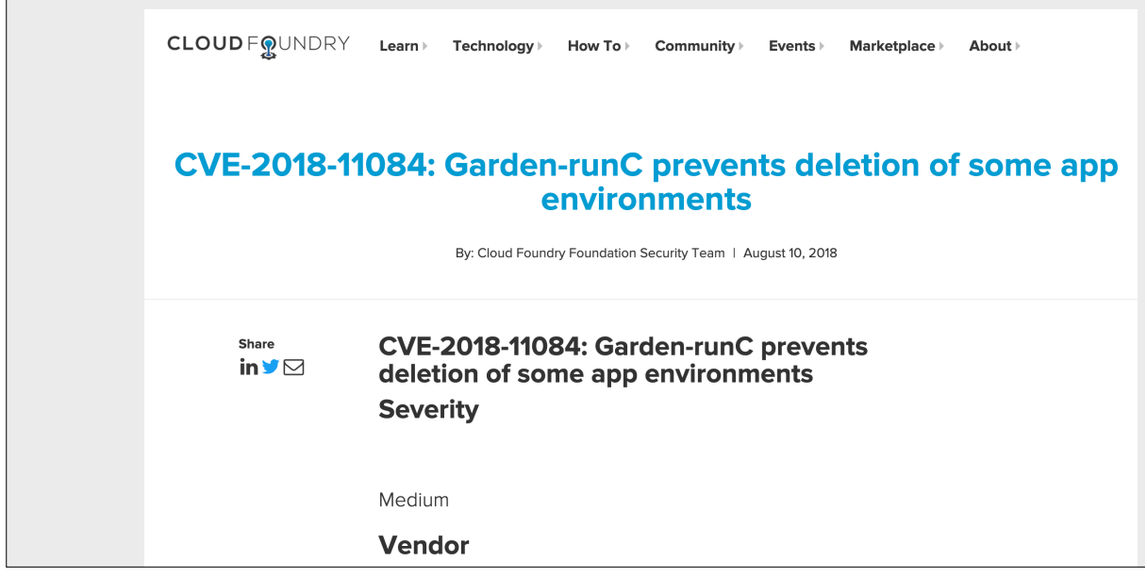
Lets dive into the problem that came from the bug report.

The problem was that long enough file paths could not be removed.
What do I mean by long? Well, very long. This long. Longer than 4000 characters on the system we were testing.

We quite quickly noticed that it wasn’t our code that was failing, but it was Golangs code. Specifically the RemoveAll function in the os package.

Being honest, paths this long are quite rare. They would probably either have to be created by accident, or with malicious intent..

An Edge Case But..



This is why in Cloud Foundry's eyes, the bug was even promoted to a CVE - Malicious uses might be able to cause denial of service by creating many undestroyable containers.

Our team therefore had a need to work out a fix something. And this was version 1.

The Solution

```
func rmRf(path string) error {
        output, err := exec.Command("rm", "-Rf", path).CombinedOutput()
        if err != nil {
                return fmt.Errorf("rm -rf %s failed: %s; original error: %s", path, output, err.Error())
        }

        return nil
}
```

- Simple Solution
- Uses a well-known and trusted tool

Anyone who has used a command prompt will probably have stumbled upon the remove tool. And anyone who has worked on a production server is probably aware of the -Rf flags, that force the recursive removal of an entire directory.

Our first solution was to use this tool. The command function, which does the shelling out, runs rm -rf as if we were using the command line. In some ways, why not? The solution is simple, small and uses an extremely well known and trusted tool, which can remove the long path files.

We committed our code, congratulated ourselves on a good job, and called it a day.

Thanks for coming to my talk :)

Well that could have been the end of this talk. But it wasn’t.

```go
func rmRf(path string) error {
        output, err := exec.Command("rm", "-Rf", path).CombinedOutput()
        if err != nil {
                return fmt.Errorf("rm -rf %s failed: %s; original error: %s", path, output, err.Error())
        }

        return nil
}
```

How many of you think that shelling out to rm -rf was a good solution? How many not?

The Problem With the Solution?

- Shelling out to a binary can have undesired consequences..
  - Ugly!
  - Windows anyone?
- What was the actual problem?

Dig into `os`

os.RemoveAll()

```go
func RemoveAll(path string) error {
        // Simple case: if Remove works, we're done.
        err := Remove(path)
        if err == nil || IsNotExist(err) {
                return nil
        }

        // Otherwise, is this a directory we need to recurse into?
        dir, serr := Lstat(path)
        if serr != nil {
                if serr, ok := serr.(*PathError); ok && (IsNotExist(serr.Err) || serr.Err == syscall.ENOTDIR) {
                        return nil
                }
                return serr
        }
        if !dir.IsDir() {
                // Not a directory; return the error from Remove.
                return err
        }

        // Remove contents & return first error.
```

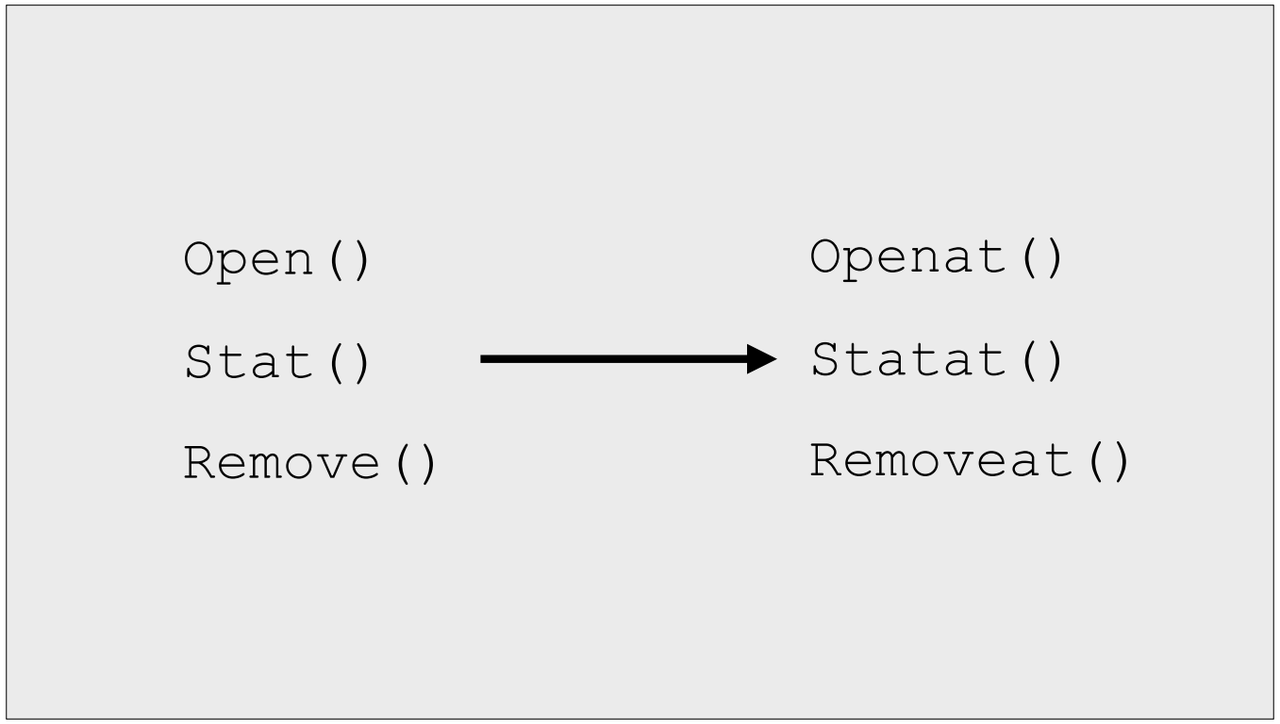
Failed already here

“file name too long”

Now this is what the beginning of Go’s RemoveAll function used to look like.

The entire thing is around 100 lines of recursive removal code.
What amazed me was that this code is completely platform independent.
In some ways you might hope that removing files could be the same on all systems, but in the end it is only because the abstractions in golang are powerful enough.

We found that the function fails already here for long path names. This was where the error message was coming from. The path was too long for `Lstat`



Looking a little closer, we thought that all typical file functions like open, stat and remove would fail if the path was long enough, and that their at variants should be used instead.

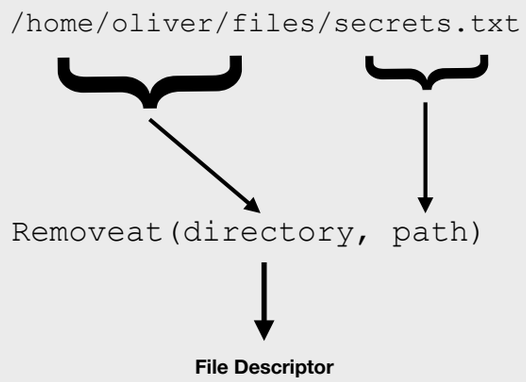
`*at()` Functions

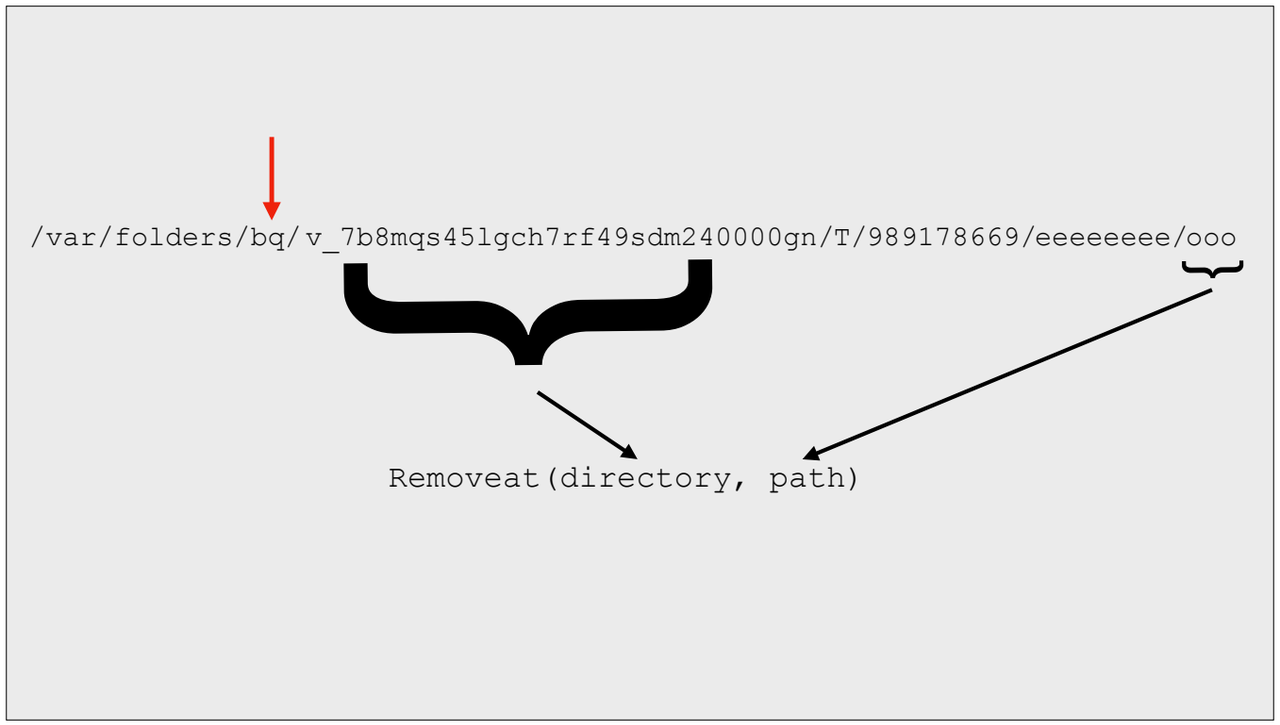
`/home/oliver/files/secrets.txt`

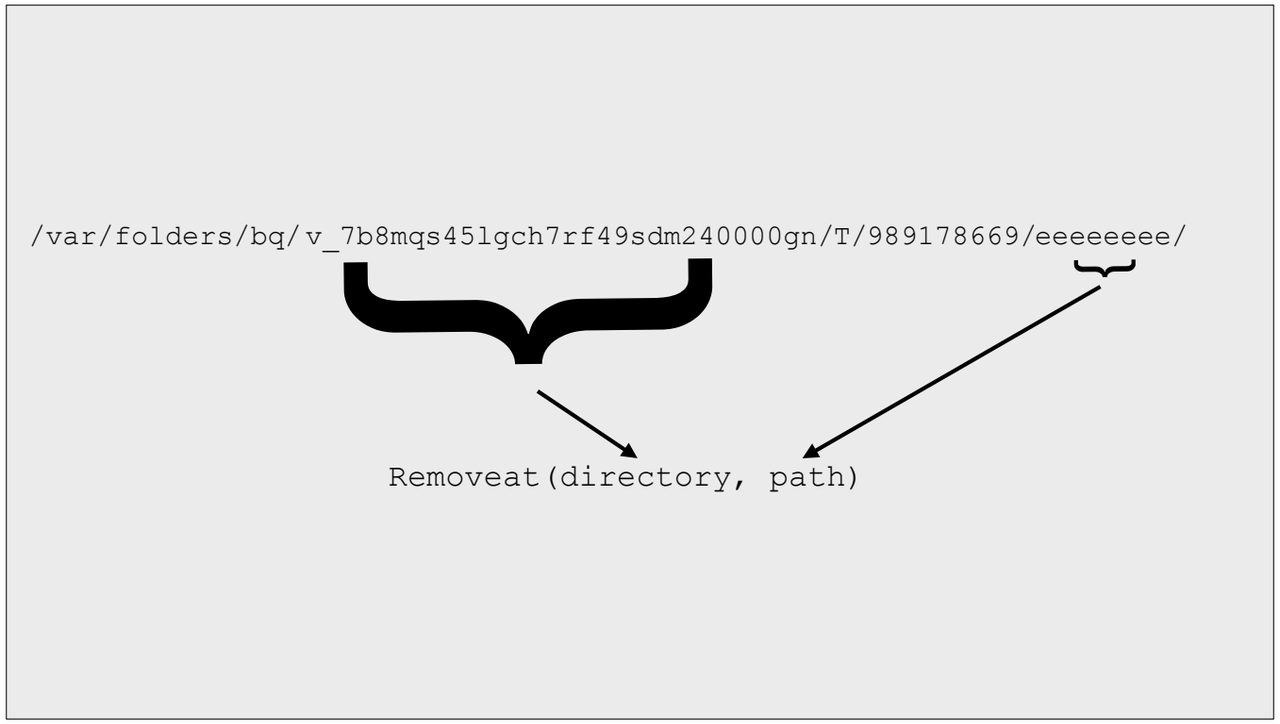
`Remove()` / `Stat()` / `Open()`

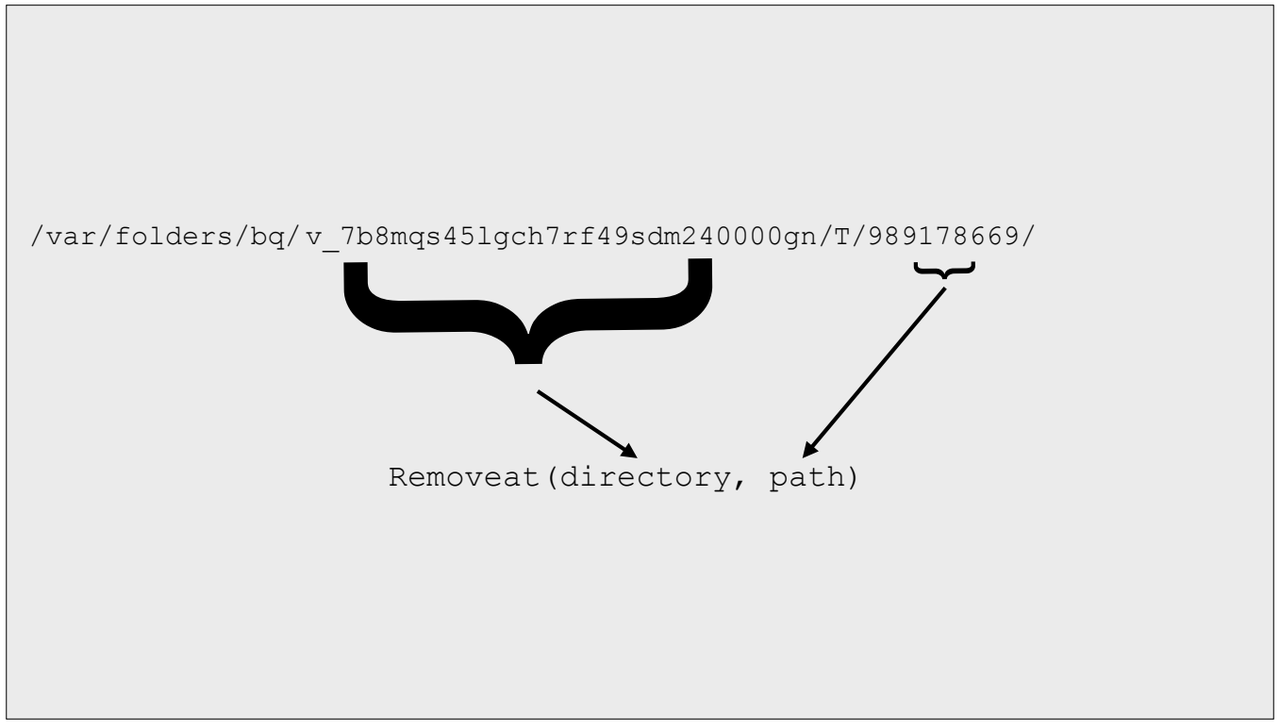
These don’t like long paths!

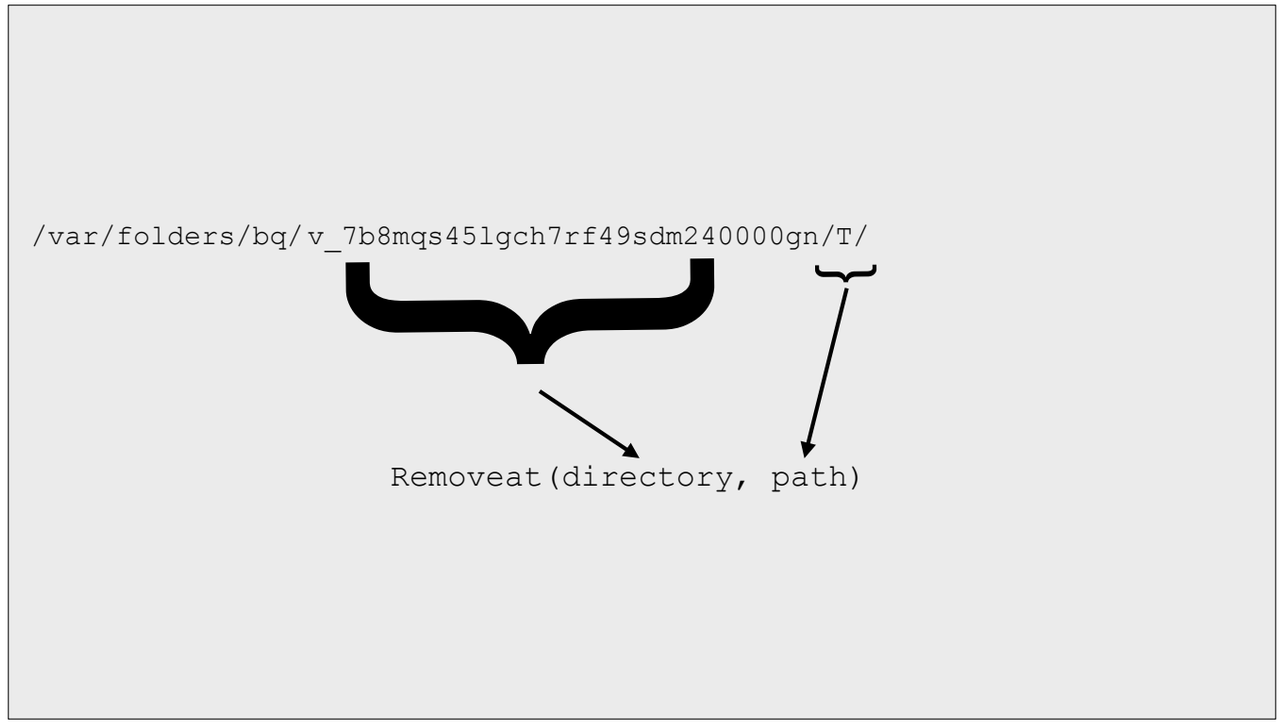
`*at()` Functions

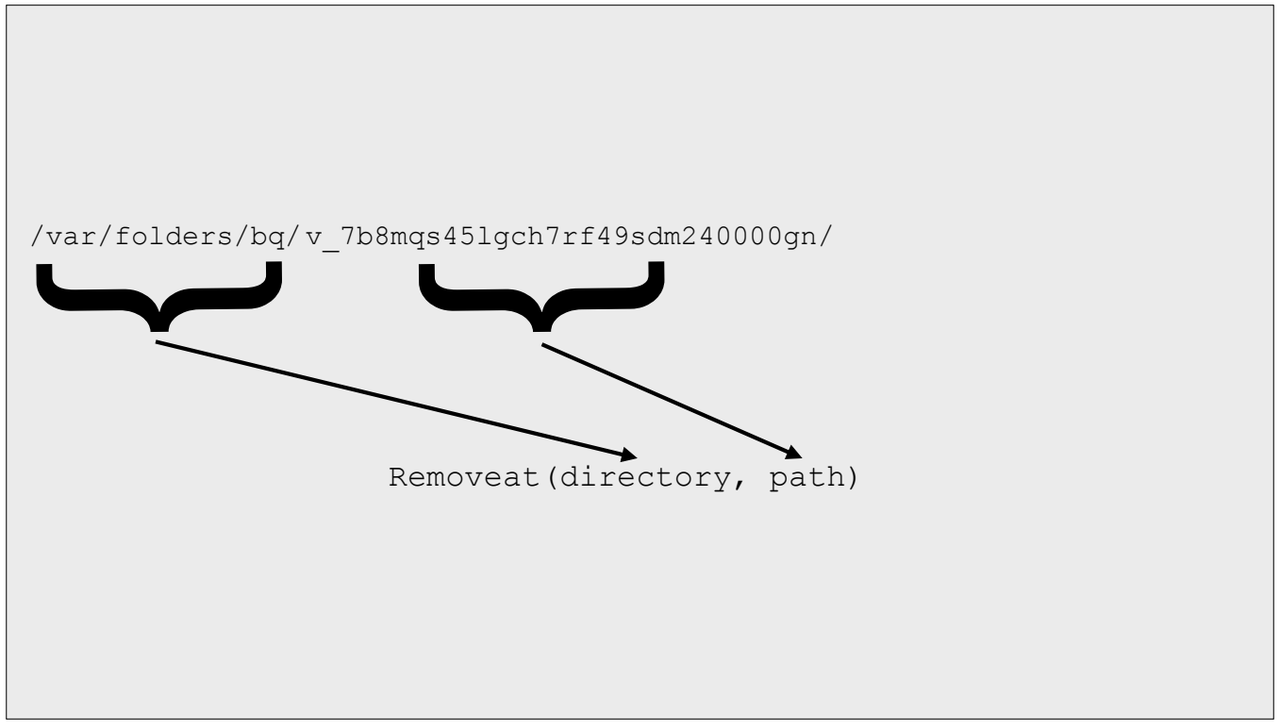


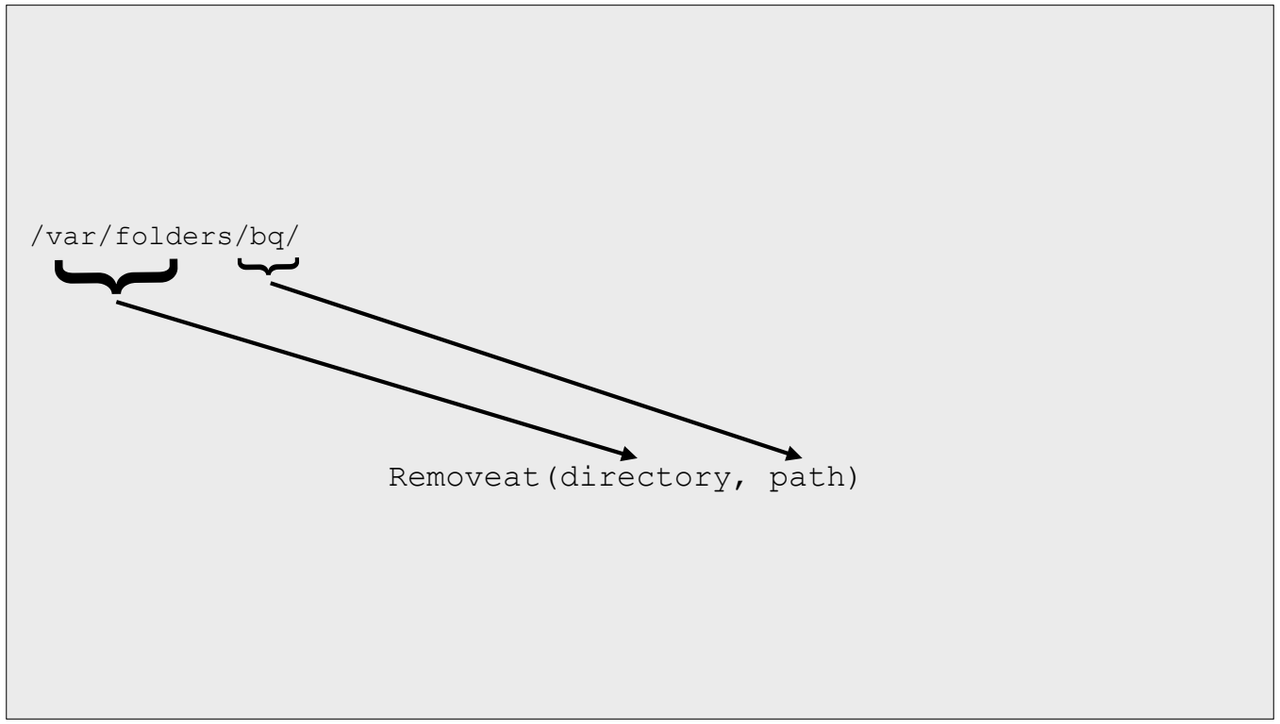

/var/folders/bq/v_7b8mqs45lgch7rf49sdm240000gn/T/989178669/eeeeeeee/ooo
Removeat(directory, path)


/var/folders/bq/v_7b8mqs45lgch7rf49sdm240000gn/T/989178669/eeeeeeee/
Removeat(directory, path)


/var/folders/bq/v_7b8mqs45lgch7rf49sdm240000gn/T/989178669/
Removeat(directory, path)


/var/folders/bq/v_7b8mqs45lgch7rf49sdm240000gn/T/
Removeat(directory, path)


/var/folders/bq/v_7b8mqs45lgch7rf49sdm240000gn/
Removeat(directory, path)


/var/folders/bq/
Removeat(directory, path)

Better Solution?

Copy Golangs `os.Removeall()` → Replace `Stat() / Remove()` with `*at()` versions

Catch: ~~Removeat()~~ takes 2 arguments: Recursion must be slightly modified

Unlinkat()

```go
func RemoveAll(path string) error {
        // What if I am removing "." or ".." -> Not allowed
        // What if I am removing "/"
        path, err := filepath.Abs(path)
        if err != nil {
                return err
        }
        parentPath := filepath.Dir(path)

        base := filepath.Base(path)

        parent, err := os.Open(parentPath)
        if os.IsNotExist(err) {
                return nil
        }
        if err != nil {
                return err
        }
        defer parent.Close()

        return removeAll(parent, base)
}
```

Prepare initial arguments

Begin Recursion

```go
func removeAll(parentFile *os.File, path string) error {
        parentFd := int(parentFile.Fd())
        err := unix.Unlinkat(parentFd, path, 0)
        if err == nil || err == unix.ENOENT {
                return nil
        }

        if err != unix.EISDIR && err != unix.EPERM {
                return err
        }

        fd, err := unix.Openat(parentFd, path, unix.O_RDONLY, 0)
        if err != nil {
                return err
        }

        file := os.NewFile(uintptr(fd), path)

        recurseErr := removeDirEntries(file)

        file.Close()

        unlinkError := unix.Unlinkat(parentFd, path, unix.AT_REMOVEDIR)
        if unlinkError == nil {
                return nil
        }
        if recurseErr != nil {
                return recurseErr
        }
        return unlinkError
}
```

File case

Directory case: remove everything inside

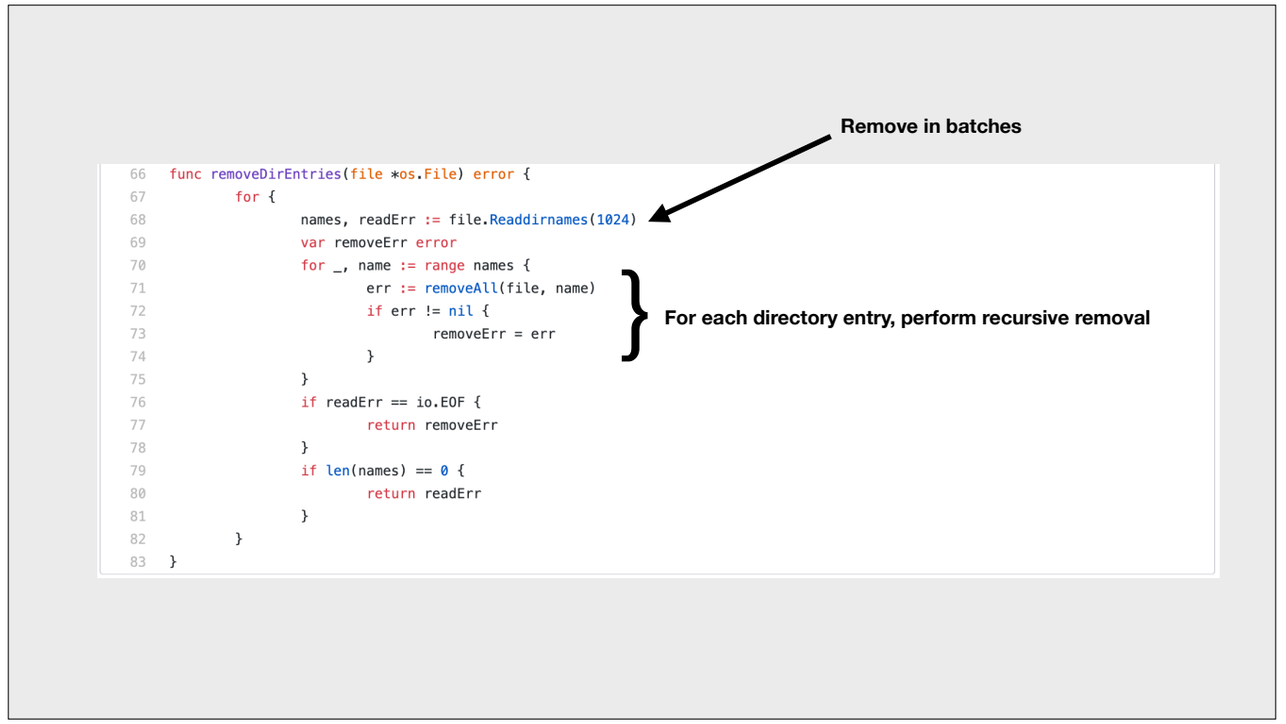
Once the directory is empty, remove it

```
34  func removeAll(parentFile *os.File, path string) error {
35          parentFd := int(parentFile.Fd())
36          err := unix.Unlinkat(parentFd, path, 0)
37          if err == nil || err == unix.ENOENT {
38                  return nil
39          }
```

} File case

```
49
50          file := os.NewFile(uintptr(fd), path)
51
52          recurseErr := removeDirEntries(file)
53
54          file.Close()
55
```

Directory case: remove everything inside




```
66  func removeDirEntries(file *os.File) error {
67          for {
68                  names, readErr := file.Readdirnames(1024)
69                  var removeErr error
70                  for _, name := range names {
71                          err := removeAll(file, name)
72                          if err != nil {
73                                  removeErr = err
74                          }
75                  }
76                  if readErr == io.EOF {
77                          return removeErr
78                  }
79                  if len(names) == 0 {
80                          return readErr
81                  }
82          }
83  }
```

```
56          unlinkError := unix.Unlinkat(parentFd, path, unix.AT_REMOVEDIR)
57          if unlinkError == nil {
58                  return nil
59          }
60          if recurseErr != nil {
61                  return recurseErr
62          }
63          return unlinkError
```

Once the directory is empty, remove it

We committed our code, congratulated ourselves on a job well done, and went for beers

This could have been the end of our story too. We’d learnt something about linux and fixed the problem in a satisfactory way. Job done?

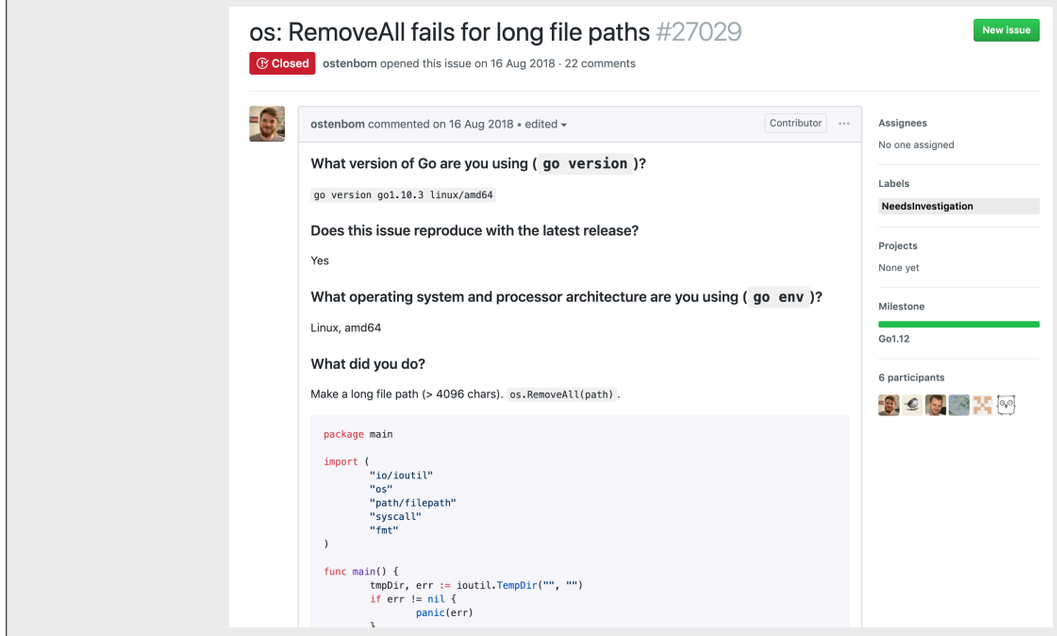
Instead we thought this would be a great opportunity to give something back. Maybe we weren't the only ones who could be DoS'ed by this, maybe we love removing code, maybe we thought the Go we'd come to love deserved to learn from our experiences. It had taken us a lot of effort to get to the point where we were able to understand and solve this, and not sharing that might cause someone else to have to spend that effort too.

We decided to set sail on our contribution voyage

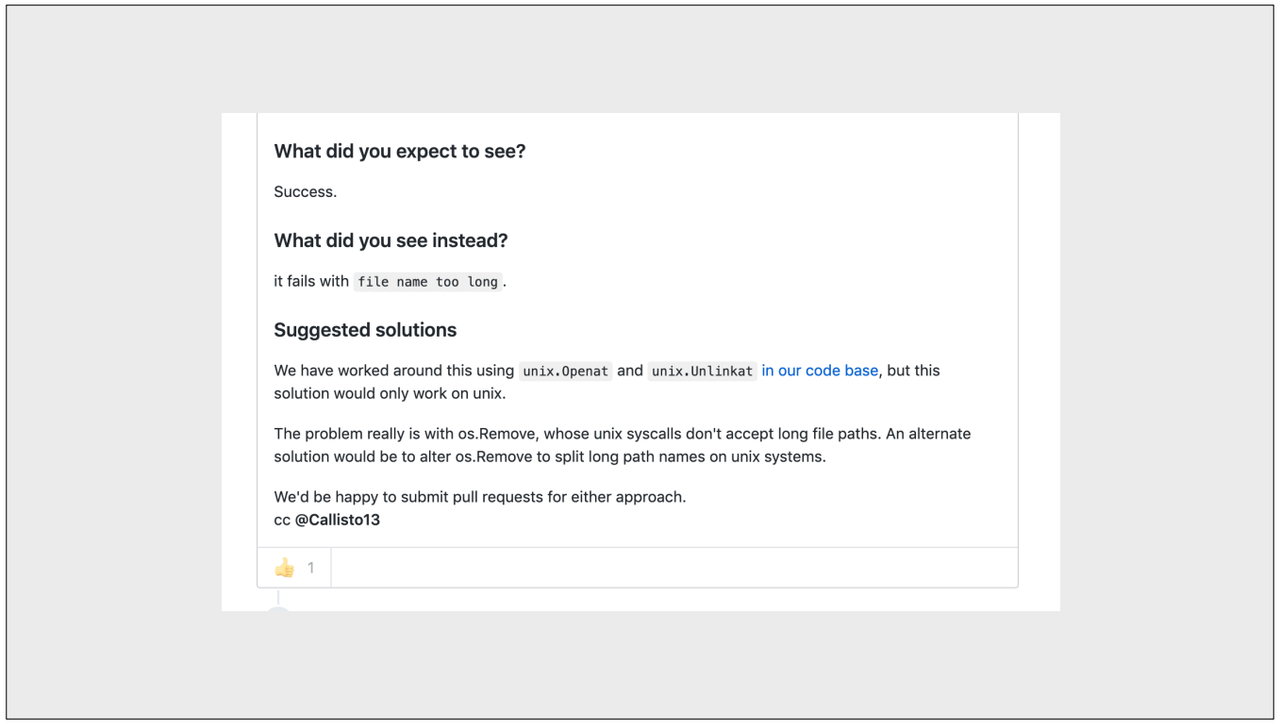
The truth was that at this point in time, many of the engineers on our team had been a part of creating our solution. To the best of our knowledge we had build a ship-shape solution, ready with polished tested code.

We imagined our voyage being a glorious entrance in to port, blessing the go world with our magnificent code and celebrating our contribution accomplishment.

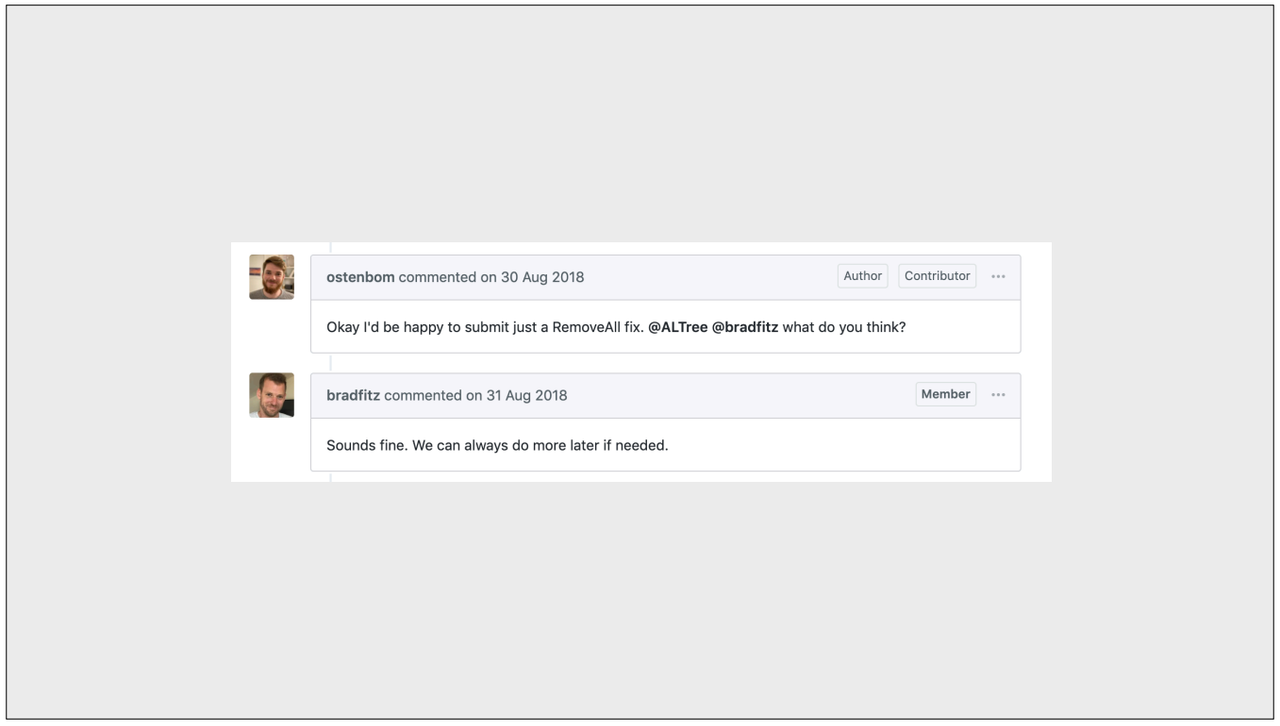
The truth is that land that you can see in the distance can be much further away than it may appear.



We began our contribution voyage by opening an issue. It contained a code snippet that could accurately recreate the problem in go, easy for the maintainers to verify.



We highlighted the problem, and suggested a potential solution.



After some further conversation with the maintainers, we got the all ok to draft a solution for golang

This is where I realised that I felt more like this monkey on our ship, rather than a sea-worthy captain.

There were plenty of completely new concepts and systems to get to grips with, and then some problems beyond that

Getting going

- The golang test suite
- Cross platform compatibility
- Non existent system calls

Besides from getting to grips with the basics of contribution like sorting out CLA’s and becoming familiar with googles Gerrit system for commits, we had three large hurdles to overcome.

The Golang Test Suite

- Get it working
- Understand how tests are made
- Make one or more of our own
- Focus a test anyone?

Cross-Platform compatibility

- Previous RemoveAll() - simple, platform independent
- *at syscall version is unix specific
- Use the more general one if you can, make it more specific if you have to
    - Part of language maturity
    - // +build !linux …

What was nice about the initial solution. Simply a recursive remove function - Not much necessarily unix with this solution.
But.. this solution wasn’t working, we had to make a *at syscall version - something unix specific.
Here the rule of thumb is to start with a general solution and then to make it more specific as problems arise and the language matures.
Some platform independent solutions in golang may stand the test of time. Part of creating a better language is working out where that is not the case - here, specific, nuanced solutions should be tailor made, which is what we had to do.
The way this is done in golang is by using build flags, which are comments at the top of source files describing which systems to compile this code on. This lets you have different RemoveAll functions for windows and unix.

Non-existent syscalls

- Stat, Open, Remove are system calls under the hood
- A system call is…
  - Special instruction
  - Number + arguments
  - Number different on each system
- *at not in core Golang - /x/ not an option

Our final problem

Non-existent syscalls

Linux

Darwin

FreeBSD

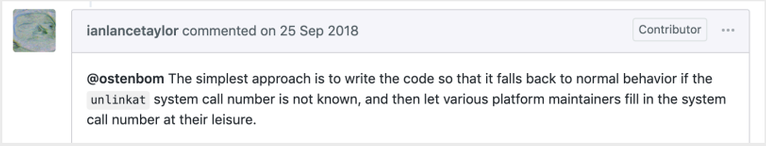
OpenBSD

NetBSD

Dragonfly

Suddenly I felt like lying on the floor. Our seemingly simple solution continued to grow - where would we see the end of it?!

This is where we have to ask ourselves the question - How deep do you have to go? How should you do? How much do you have to do?

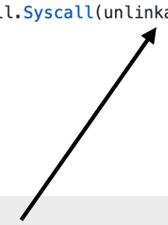


The big learning point here is that there is no need to do everything at once.

If anything doing everything at once will probably make your contribution much less likely to succeed. It could introduce more errors, or drag on, or your yourself might even get tired of it!

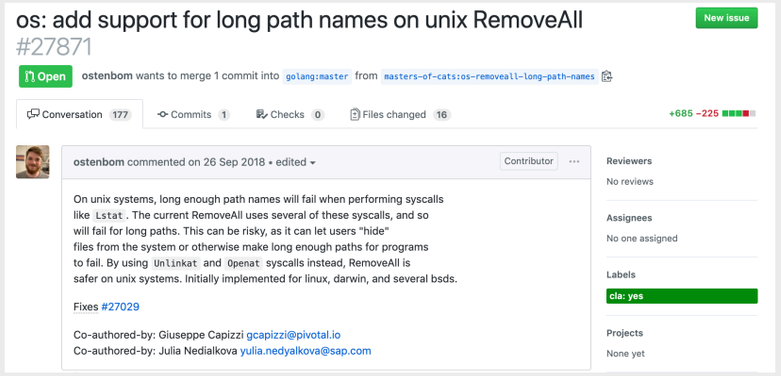
Take it in manageable chunks if possible, and let the community help you fill in the gaps!

```go
14  func Unlinkat(dirfd int, path string, flags int) error {
15          var p *byte
16          p, err := syscall.BytePtrFromString(path)
17          if err != nil {
18                  return err
19          }
20
21          _, _, errno := syscall.Syscall(unlinkatTrap, uintptr(dirfd), uintptr(unsafe.Pointer(p)), uintptr(flags))
22          if errno != 0 {
23                  return errno
24          }
25
26          return nil
27  }
```



Constant that depends on the system

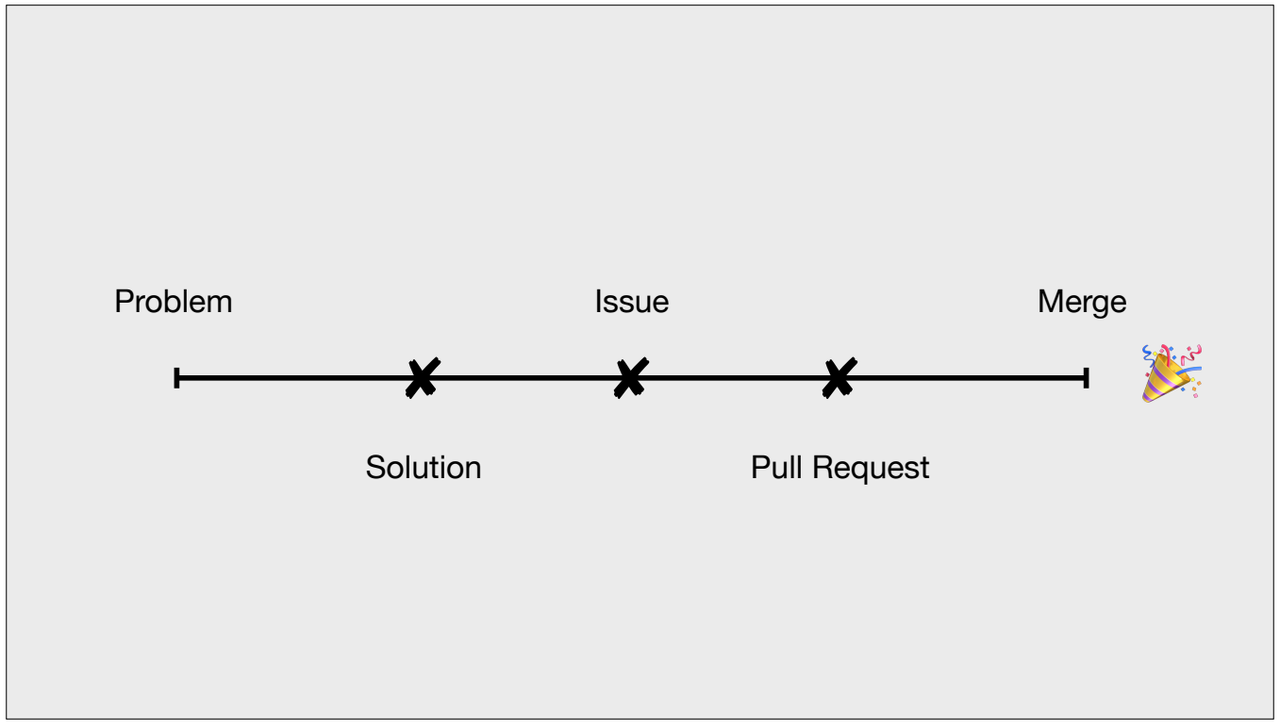
Also needed Openat, Fstatat…



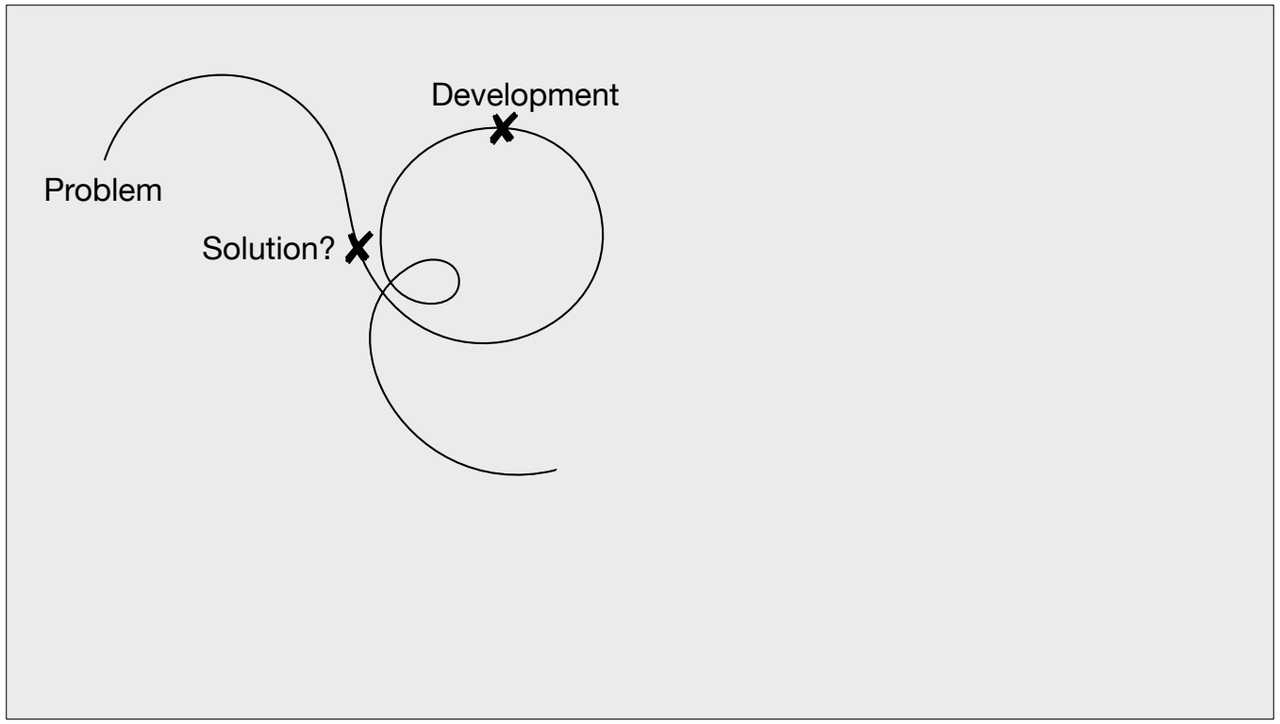
We implemented a few basic systems: linux, Darwin and a BSD or two.

Now it really felt like we were close to port and that just mooring up would be the last step

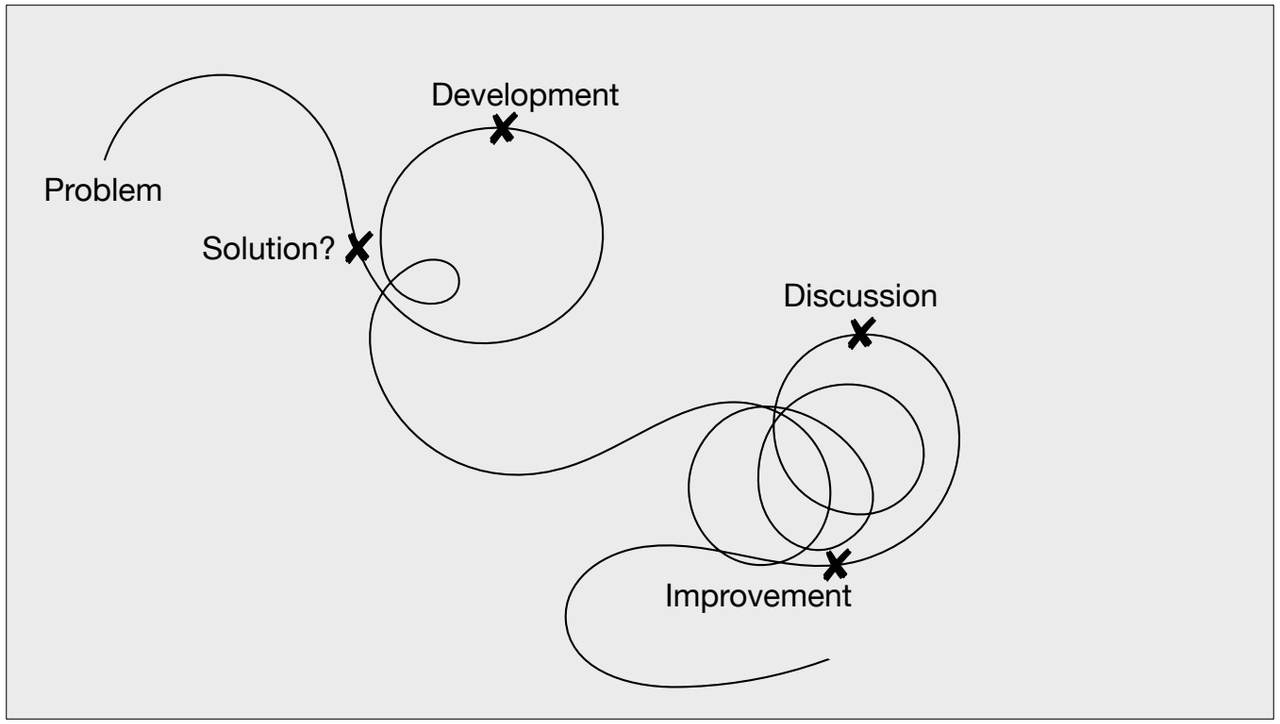
But land in the distance still feels closer than it actually is.


Problem
Issue
Merge
Solution
Pull Request

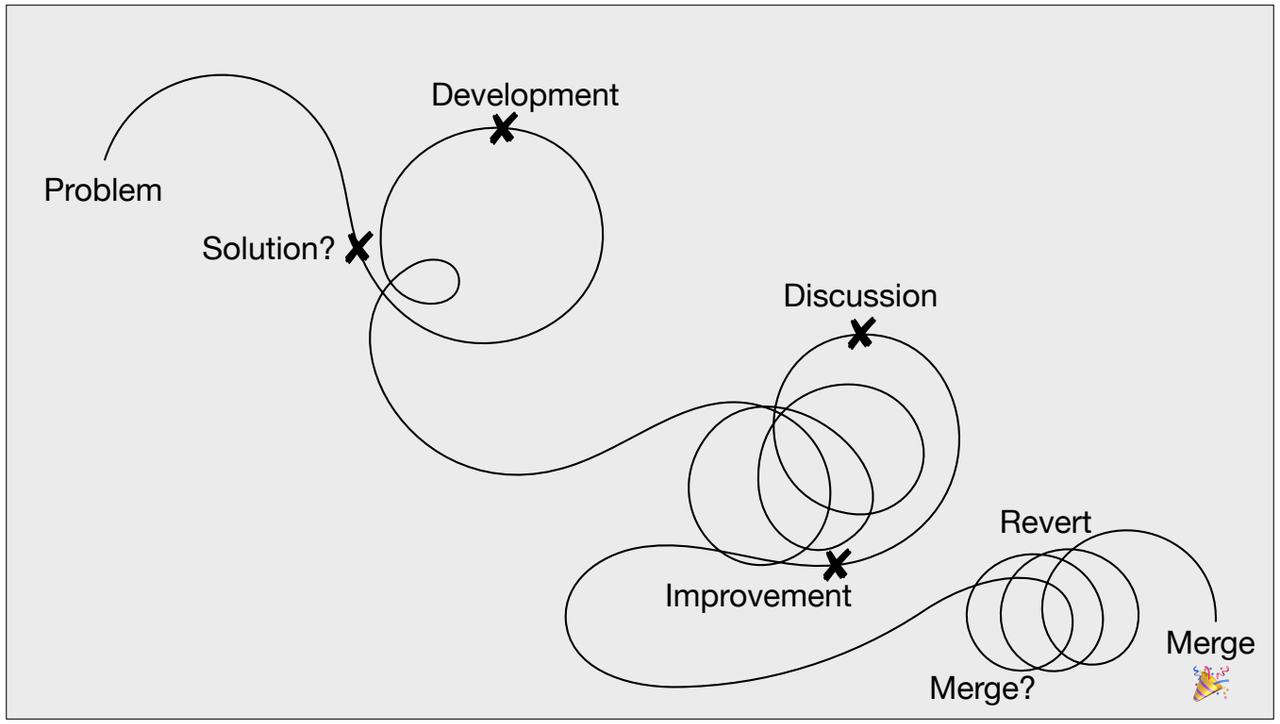

Problem



More like 3 solutions and plenty of unforeseen problems



10 messages to decide the name of a file.. Around 200 messages total

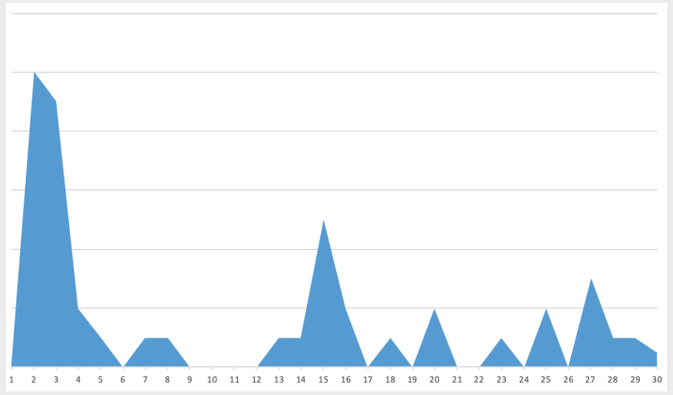


Even once we had got it merged it wasn’t the end. We managed to break the test suite, so had to revert, fix and merge again

Even then that wasn’t the end of the story! There have been several pull requests since not just to add platform support, but to fix other minor errors.

It was tough because..

- Slow pace (at least in europe)



***Not based on actual data**

And it’s not like all of this work could be done at once. This is a demonstration of how much you might have to work on a contribution. Although the initial bulk of work created that first solution, there were then little peaks for weeks afterwards to improve the code.

It was tough because..

- Slow feedback pace (at least in europe)
- Freezing deadline
- Extremely high standards
- Didn’t turn out as expected

High standards for good reason. Go 1.12 which this fix was included in has already shipped all around the globe to all sorts of systems. This function is probably going to run millions if not billions of times in the next year - getting it perfectly right first time isn’t always possible, but everything around golang contribution is in place to get pretty damned close. The time cost is probably neccesary - the language is at stake.

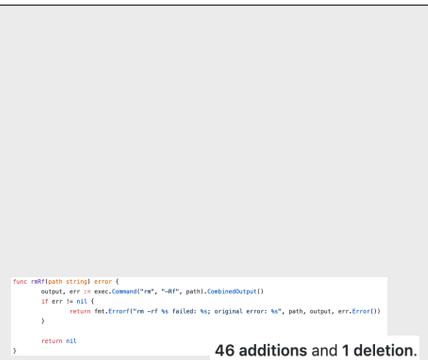
It didn’t end up looking the way I thought it would
By the time it was in, I was sad that it wasn’t as “clean” as I would have liked it to be - due to compatibility reasons. The way it looked was very different to how it was in the beginning

But!..

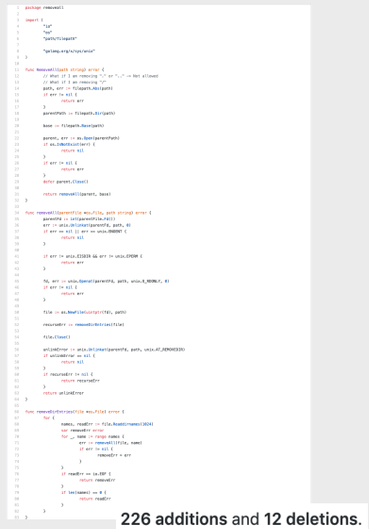
Rewards

- Became knowledgeable in:
    - The os package
    - Unix system calls
    - High quality Golang

It didn’t matter much that it didn’t look the same or I was up a few nights or that the semantics got really annoying at times. I was taking tips from the core maintainers of the language, and they were spending a lot of time getting back quickly with detailed and constructive advice - it didn’t matter that I was some student on the other side of the planet or that I was a rookie who wrote his first line of Golang 3 months earlier. It was invaluable. Contributing helped me understand the language. It helped me understand the package I was working in - os and a thing or two about unix syscalls. This gave me the confidence to do a low-level, golang based masters project that allowed one process to run arbitrarily run system calls inside of another. It also taught me a thing or two about high-quality Golang.



rm -rf



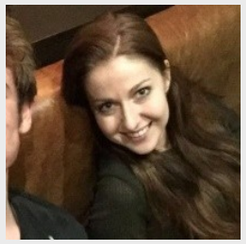
Our solution

Golang ready

To round off I want to look at the evolution of the solution to that problem that we started out with.
In our case we had to fix it asap. Quick and a bit dirty was ok. The bug was critical so we got an iteration out, and followed up
Our proper solution that we then conjured up was vastly different from the first. The one that eventually got merged was very different to that one still.
That’s ok - we’re not perfect and a lot of people and work went into refining it. Here smaller is not necessarily better.
What I want you to take from this talk is that so much stuff came from that initial bug report. We could have said “you know what, shelling out to rm -rf is fine”, with the mindset that someone else would fix it. But we didn’t. NOT NECCESARILY SMALL & SIMPLE
Which one works for your team? Your story may be completely different, and you may not have the money or time to embark on this journey, but I can at least assure you which solution you’ll learn the most from.

Thanks To

Yulia



Claudia

Giuseppe

Tom

Julz

Danail

Georgi

Of course, a load of that credit also has to go to Yulia, Tom, Danail, Giuseppe and Julz for the help along the way.

Thanks to all the time the contributors spend giving friendly feedback. I really have to thank them.

Round of applause fore the community that helps contribute to go.

How Deep Do You *Go*?

Contributing to the os Package

Oliver Stenbom

July 25th
Gophercon 2019

Let's talk!

Thanks for staying awake so late into the afternoon. Feel free to talk to me about contribution, syscalls in golang or serverless containers. I'm Oliver Stenbom, you stay classy Gophercon.