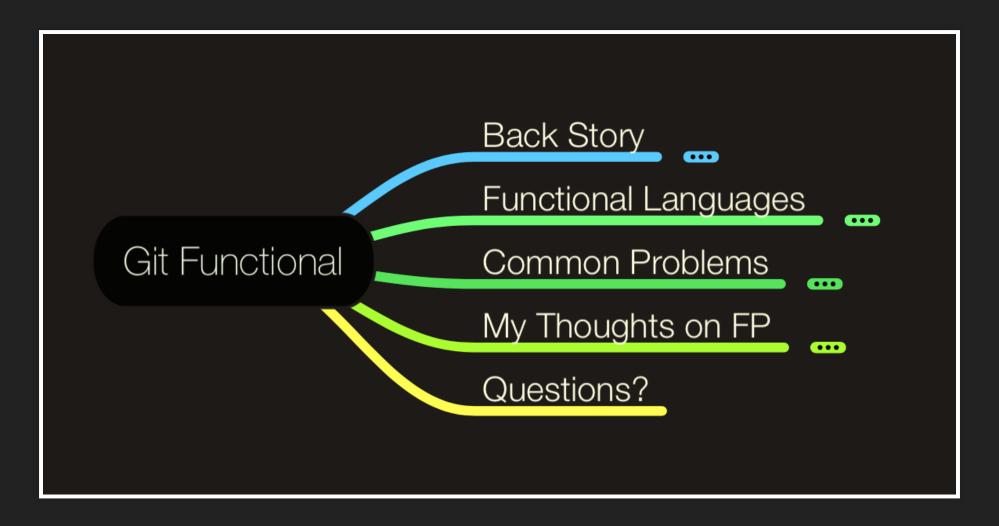
# GIT FUNCTIONAL

A memory dump of Lambda Squared and some words on Functional Programming.

PRESENTER: AMOS GARNER

## **SUMMARY**



## **BACKSTORY**

In the beginning there was suffering.
Then there was Haskell, and the
suffering continued.

Do not try to change the state; that's impossible. Instead only try to realize the truth: There is no state.

## **DISCLAIMER:**

I went to one conference and did some googeling. This does not make me an expert.

Opinionated... sure, but not an expert.
So be sure to take this talk with a whole cup of salt.

## TERMINOLOGY: 1/2

#### **Functor**

Set of rules that are expected from a function's I/O.

#### Lambda

A function that can be treated like a value.

#### **Monaid**

A function that takes in a input type and returns that same type.

#### Monad

An object containing a return and a chain.

## TERMINOLOGY: 2/2

#### **Mutability**

Determins if data can or cannot change.

#### **Side Effects**

Code in this function interacts with an external muttable source.

#### **Purity**

The return value must only rely on it's inputs without side-effects.

#### Currying

Converting a function with multiple arguments into a one-at-a-time function.

## **FUNCTIONAL LANGUAGES**

- Haskell
- Lisp
- Erlang
- Kotlin
- PHP, JS, Python \*\*

Can be incorporated into any language. I like to call this being Objectively Functional or Functionally Objective.

## COMMON PROBLEMS W/ FUNCTIONAL SOLUTIONS

## WHEN CODE IS HARD TO TEST

Remove complexity to get to core logic.

```
async function logStuffToConsole(ids){
    url = '/getStuff?ids={join(',',ids)}'; //
    stuff = await fetchJSON(url); //
    foreach(stuff as item){
        details = 'Thing: {item.id}, created on {item.date}';
        console.log(details); //
    }
}
```

```
function logStuffToConsole(stuff){
    details = [];
    foreach(stuff as item){
        push(details, getDetails(item));
    }
    return details;
}
```

## WHEN CODE IS HARD TO GROK

Use declarative coding to improve code consistency.

```
foreach(stuff as item){
    push(details, getDetails(item));
}

details = stuff.map(item, {
    return getDetails(item);
})
```

## HELP, MY CODE IS BREAKING

Use Immutable Data to eliminate code fragility.

```
class Thing{
    constructor(id, stuff){
        this.id = id;
        this.stuff = Object.freeze(stuff)
    }
}
```

```
thing1 = new Thing(1, ['One', 'Two']);
thing1.stuff = ['Three', 'Four']
```

```
thing2 = new Thing(thing1.id, append(thing1.stuff, 'Three'));
```

# MY THOUGHTS ON FUNCTIONAL PROGRAMMING

- Removes state from the equation
- Improves code flexibility
- Removes some code cruft
- Gets confusing in its ceremony
- Definitely some things we can use here

# ANY QUESTIONS?

Presented By: Amos Garner @BitFiendCoder

My Other Talks:

Github.com/AmosGarner/Developer\_Presentations