

ADAPTIVE *INTENT*-BASED *CLI STATE MACHINES*

@swyx



oclif conf

May 2019

rupa/z

“Frecency”

Jump to a conversation `tab` or `↑ ↓` to navigate `↵` to select `esc` to dismiss

Matt

- Matt Bylander mattb
- Matt Haughey mathowie
- Matt Heist matt
- Matt Hodgins mhodgins
- Matt Kump kump

<https://slack.engineering/a-faster-smarter-quick-switcher-77cbc193cb60>

ADAPTIVE
INTENT-BASED
CLI STATE MACHINES

RIDICULOUSLY
OVER-ENGINEERED
COMMAND LINE APPS

PART I

NETLIFY

NETLIFY CLI

NETLIFY DEV

DISCLAIMER

THIS IS MOSTLY COLLEAGUES' WORK



David Wells



Bret Comnes

JS • Go • Swift • Crypto • P2P



Ryan Neal

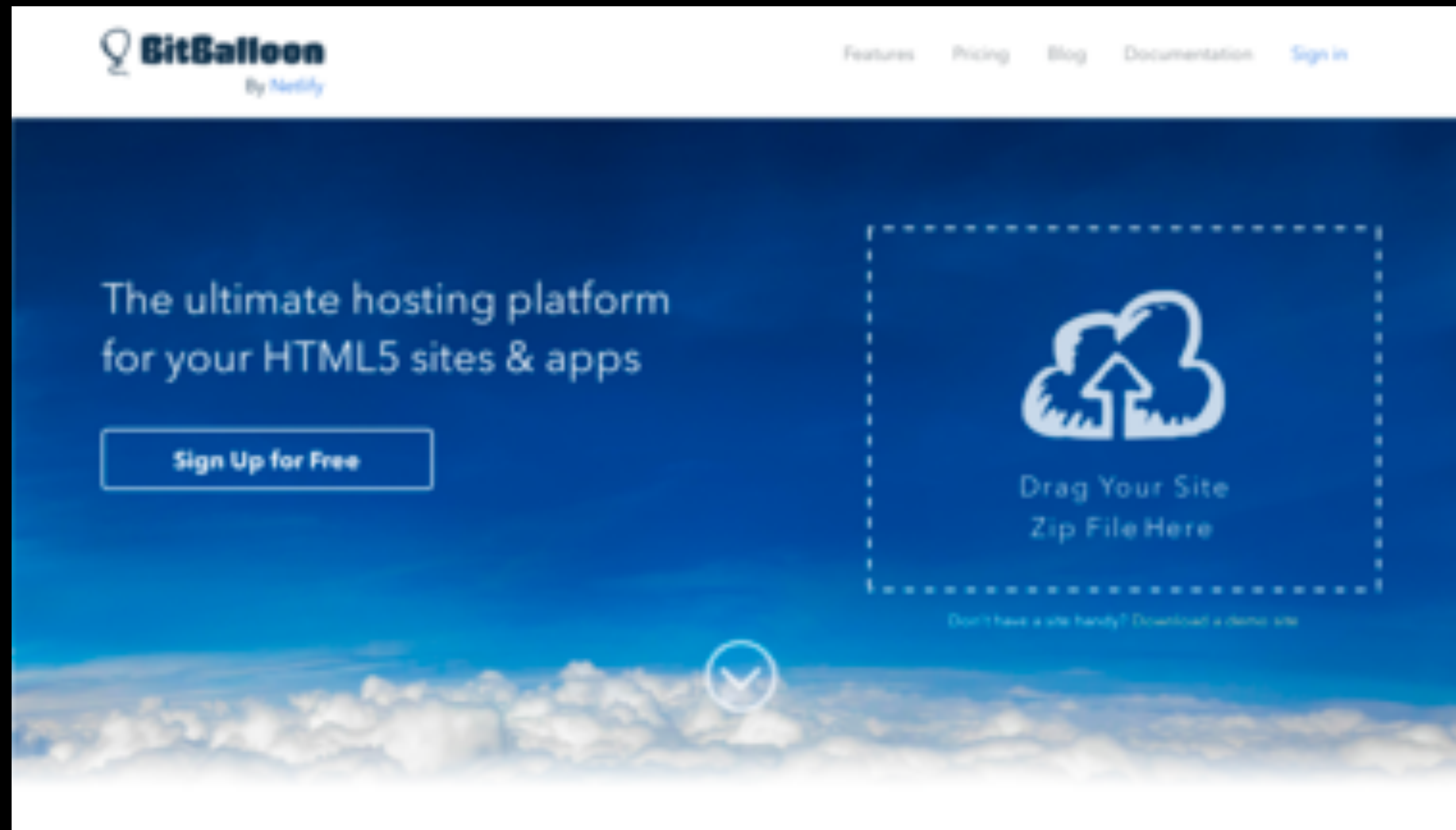


Matt Biilmann



David Calavera

BitBalloon



StaticGen.com




StaticGen

A List of Static Site Generators for JAMstack Sites




[About](#) [Contribute](#) [About JAMstack](#) [Need a Static CMS?](#)

Filter

Any Language  Any Template  Any License 

Sort

GitHub stars 

Jekyll

★ 37839
+3650

📄 134
+14

🔗 8236
+701

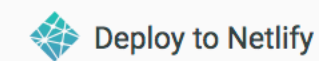
🐦 6801
+478

A simple, blog-aware, static site generator.

Languages: Ruby

Templates: Liquid

License: MIT



Next

★ 37781
+12757

📄 218
-33

🔗 4501
+2130

🐦 N/A

A framework for statically-exported React apps

Languages: JavaScript

Templates: React

License: MIT

Hugo

★ 35514
+10059

📄 368
+112

🔗 3987
+893

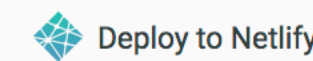
🐦 6926
+3284

A Fast and Flexible Static Site Generator.

Languages: Go

Templates: Go

License: Apache 2.0



Get started with one click!

For generators with the "Deploy to Netlify" button, you can deploy a new site from a template with one click. Get HTTPS, continuous delivery, and bring a custom domain, free of charge.

Want your own Deploy to Netlify button? [Learn more here.](#)

Gatsby

★ 35096
+13669

📄 517
-193

🔗 5087
+3046

🐦 27417
+20716

Hexo

★ 26644
+4665

📄 210
-10

🔗 3553
+485

🐦 2155
+2155

GitBook

★ 20780
+2569

📄 1002
+113

🔗 2954
+526

🐦 5233
+337

Nuxt

★ 20398
+8290

📄 148
-246

🔗 1742
+827

🐦 23982
+14205



Articles

Design & development



Books

Physical & digital books



Events

Conferences & workshops



Jobs

Find work & employees



Membership

Webinars & early-birds

Q Topics



ABOUT THE AUTHOR

Matt Biilmann has been building developer tools, content management systems and web infrastructure for more than a decade. He is co-founder and CEO of [Netlify](#), ... [More about Mathias...](#)

NOVEMBER 2, 2015 • [99 comments](#)

Why Static Site Generators Are The Next Big Thing

QUICK SUMMARY ➦ At StaticGen, our open-source directory of **static website generators**, we've kept track of more than a hundred generators for more than a year now, and we've seen both the volume and popularity of these projects take off incredibly on GitHub during that time, going from just 50 to more than 100 generators and a total of more than 100,000 stars for static website generator repositories. Influential design-focused companies such as Nest and MailChimp now use static website generators for their primary websites. Vox Media has built a whole publishing system around Middleman. Carrot, a large New York agency and part of the Vice empire, builds websites for some of the world's largest brands with its own open-source generator, Roots. And several of Google's properties, such as "A Year In Search" and Web Fundamentals, are static.

📅 18 min read

🔖 [Coding](#), [Tools](#), [Static Generators](#)

🐦 Share on [Twitter](#) or [LinkedIn](#)

[Products](#) [Pricing](#) [Docs](#) [Blog](#) [Q](#)[Contact sales](#)[Log in](#)[Sign up →](#)

Introducing: Netlify Dev. Run our entire platform right on your laptop. [Learn more →](#)

Build, deploy, and manage modern web projects



An all-in-one workflow that combines global deployment, continuous integration, and automatic HTTPS. And that's just the beginning.

[Get started for free](#)

Deploy your site in seconds*



shawn wang's team ▾

SitesDomainsMembersAudit logTeam settings



shawn wang's team ▾

Sites > trusting-lichterman-ee04c1



Deploys

Analytics

Site settings

ADD-ONS

Forms

Function

Identity

Large Media

Split Testing

Deploys

Analytics

Site settings

Build & deploy

Domain management

Access control

ADD-ONS

Forms

Functions

Identity

Large Media

Split Testing

Site settings >

Build & deploy

Build settings

Repository: [Link to a different repository →](#)

Base directory:
Directory to change to before starting a build. This is where we will look for package.json/.nvmrc/etc.

Build command:

Publish directory:

Deploy log visibility:

☒ Public logs
Anyone with a deploy detail URL will be able to access the logs.
☐ Private logs
Only site administrators will be able to access the logs.

[Learn more about common configuration directives in the docs →](#)

Save

Cancel

CLI Iterations

- V0: 🖐️ Commander.js
- V1: 🐍 Cobra
- V2: ✨ Oclif

<https://www.netlify.com/blog/2018/09/10/netlify-cli-2.0-now-in-beta-/#our-cli-journey>

Read the docs: <https://cli.netlify.com>

Support and bugs: <https://github.com/netlify/cli/issues>

Netlify **command** line tool

VERSION

netlify-cli/2.0.0-beta.2 darwin-x64 node-v10.4.1

USAGE

\$ netlify [COMMAND]

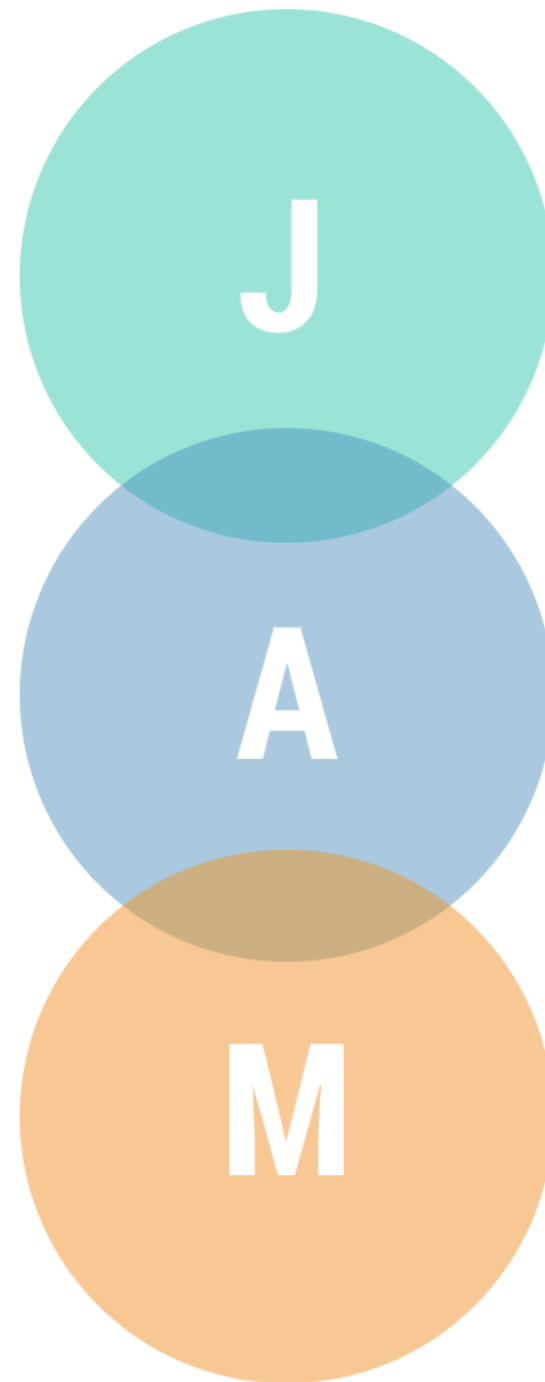
COMMANDS

deploy	Create a new deploy from the contents of a folder
init	Configure continuous deployment for a new or existing site
link	Link a local repo or project folder to an existing site on Netlify
login	Login to your Netlify account
logout	Logout of your Netlify account
open	Open settings for the site linked to the current folder
sites	Handle various site operations
status	Print status information
unlink	Unlink a local folder from a Netlify site
watch	Watch for site deploy to finish

JAMstack.org

What is the JAMstack?

Your project is built with the JAMstack if it meets three key criteria:



JavaScript

Any dynamic programming during the request/response cycle is handled by JavaScript, running entirely on the client. This could be any frontend framework, library, or even vanilla JavaScript.

APIs

All server-side processes or database actions are abstracted into reusable APIs, accessed over HTTPS with JavaScript. These can be custom-built or leverage third-party services.

Markup

Templated markup should be prebuilt at deploy time, usually using a site generator for content sites, or a build tool for web apps.

[Want to see some examples?](#)

[Products](#) [Pricing](#) [Docs](#) [Blog](#) [Q](#)[Contact sales](#)[Log in](#)[Sign up →](#)

By [Matt Biilmann](#) & [Chris Bach](#) in [News & Announcements](#) • March 20, 2018

Netlify's AWS Lambda functions bring the backend to your frontend workflow

Today we're officially releasing Functions, which make deploying serverless AWS Lambda functions on Netlify as simple as adding a file to your Git repository. We're also officially releasing Identity and Forms out of beta, so now you can add dynamic functionality to your site without setting up servers, writing server-side code, or managing multiple accounts.

Since adding these components is as easy as `git push` and manageable without a

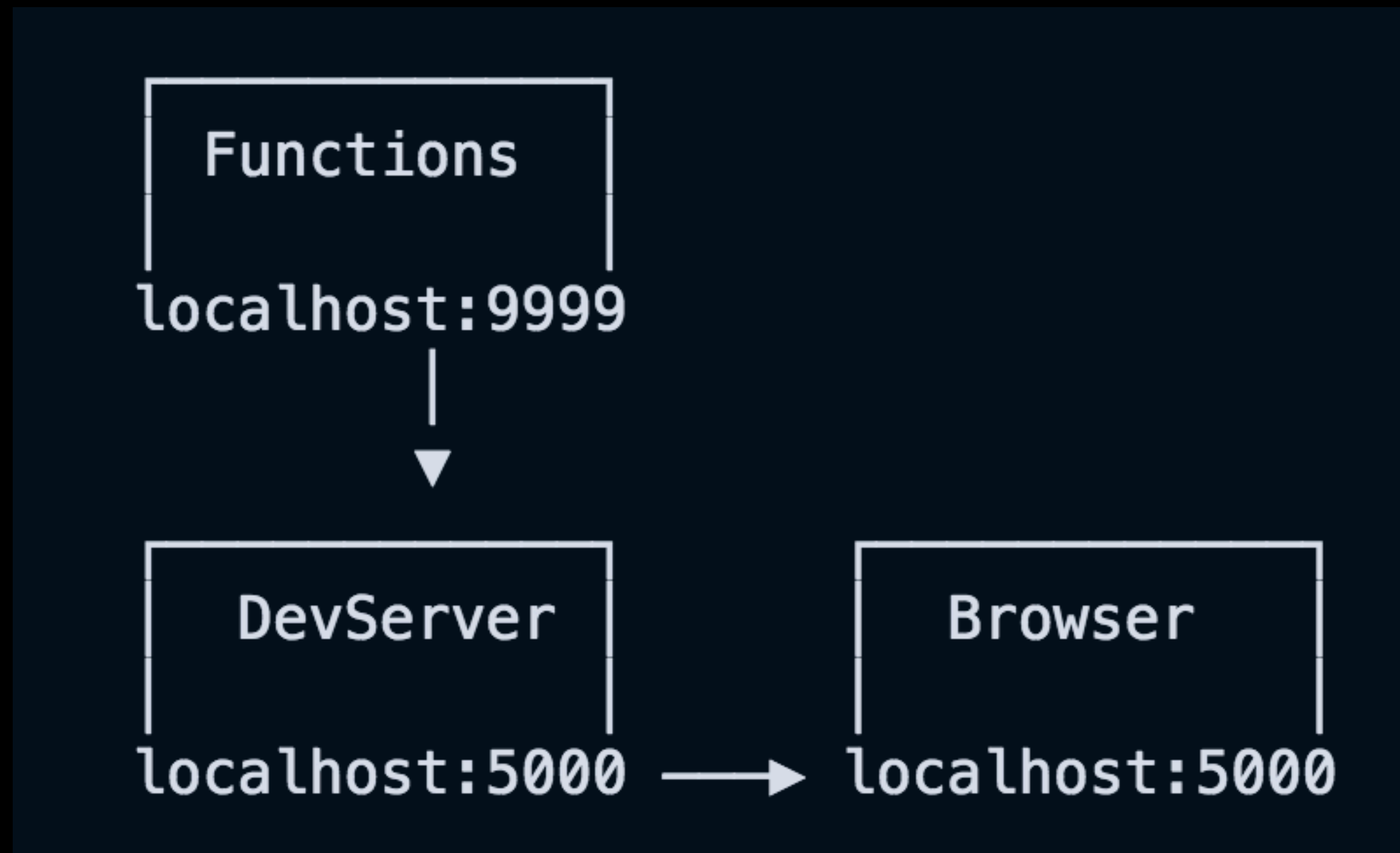
PROBLEM

LOCAL EMULATION

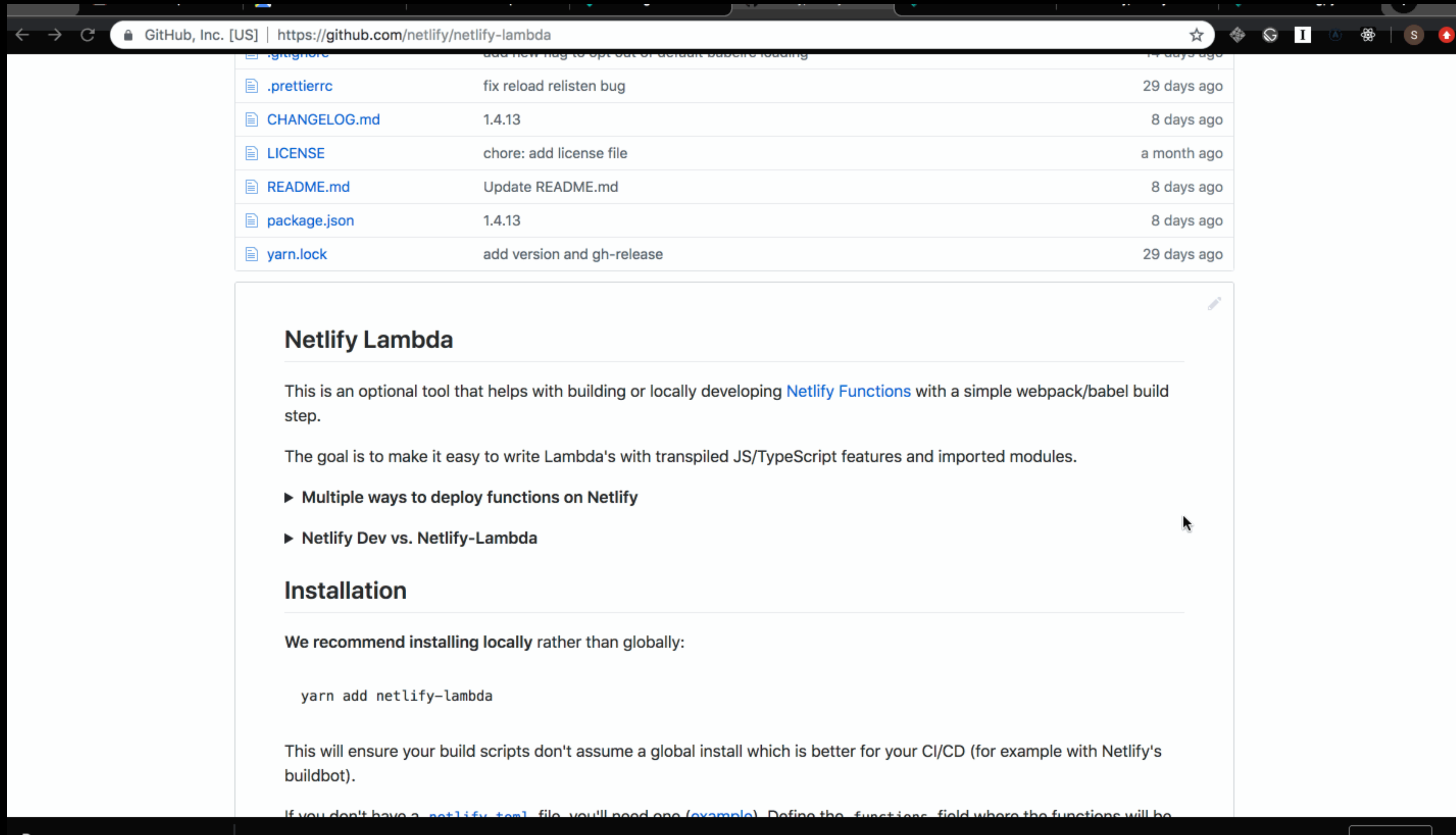
The Rise of Dev Servers



Configuring Proxies in Dev Servers



Configuring Proxies in Dev Servers



The screenshot shows the GitHub repository for Netlify Lambda. The repository name is `netlify/netlify-lambda`. The file list includes `.gitignore`, `.prettierrc`, `CHANGELOG.md`, `LICENSE`, `README.md`, `package.json`, and `yarn.lock`. The `README.md` file is selected, showing the following content:

Netlify Lambda

This is an optional tool that helps with building or locally developing [Netlify Functions](#) with a simple webpack/babel build step.

The goal is to make it easy to write Lambda's with transpiled JS/TypeScript features and imported modules.

- Multiple ways to deploy functions on Netlify
- Netlify Dev vs. Netlify-Lambda

Installation

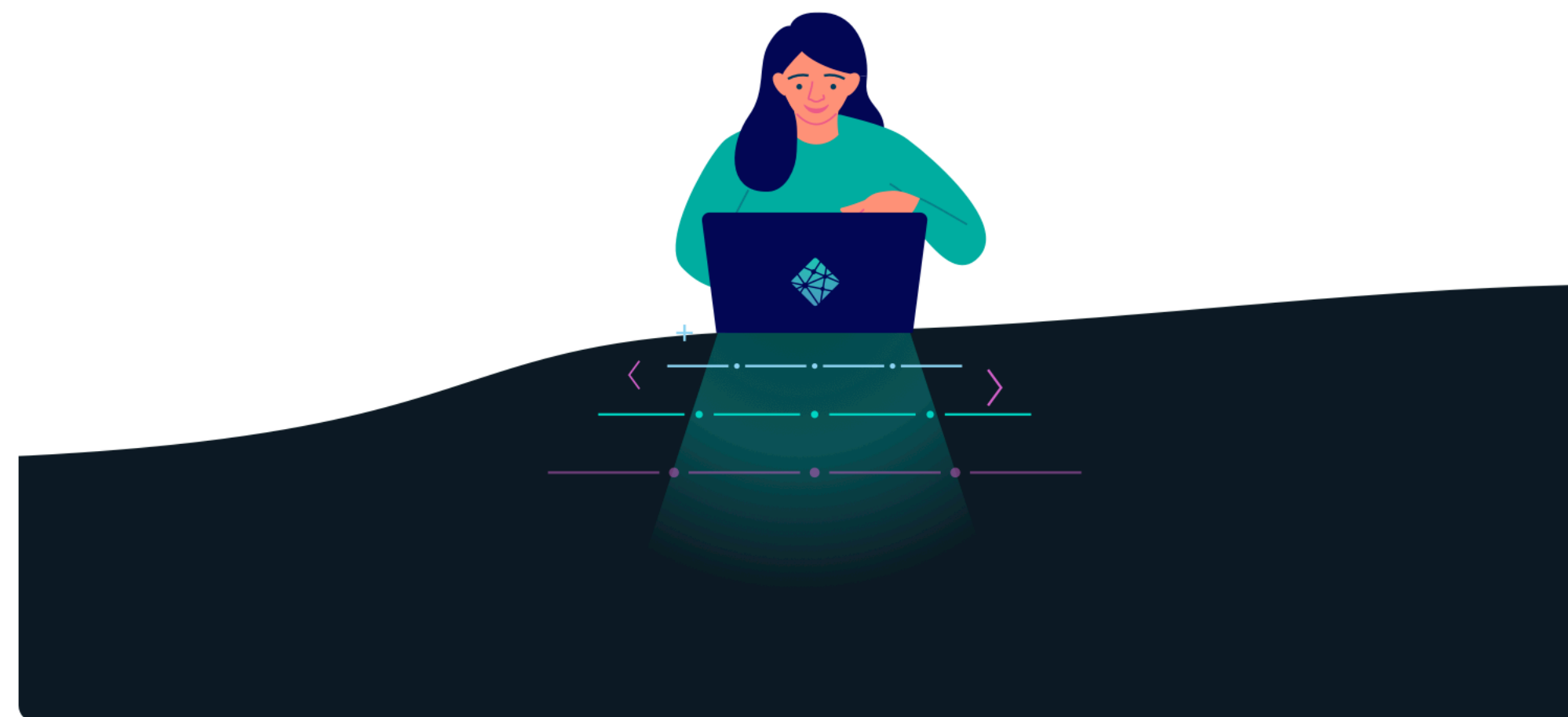
We recommend installing **locally** rather than globally:

```
yarn add netlify-lambda
```

This will ensure your build scripts don't assume a global install which is better for your CI/CD (for example with Netlify's buildbot).

If you don't have a `netlify.toml` file, you'll need one ([example](#)). Define the `functions` field where the functions will be

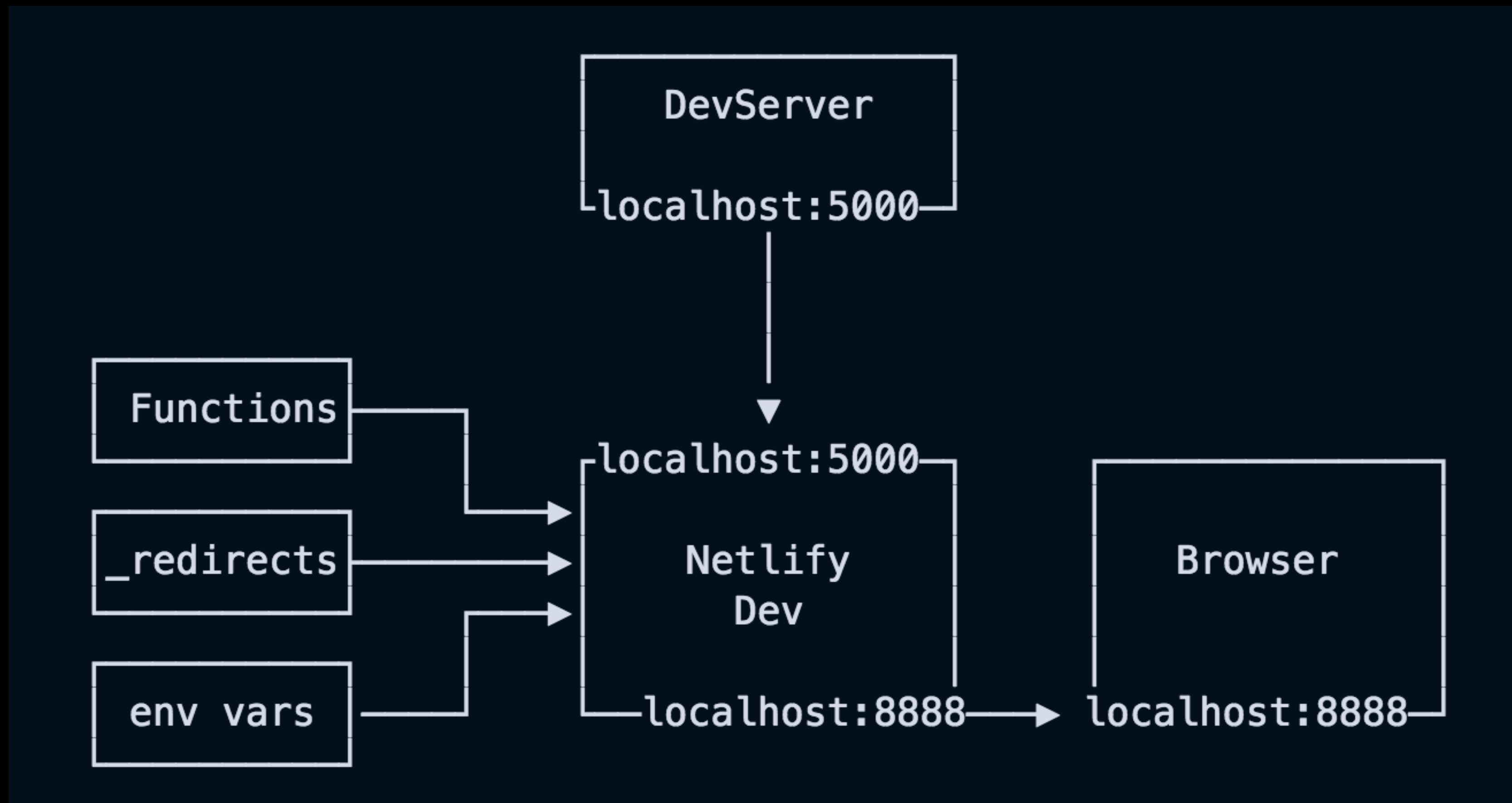
Netlify Dev — our entire platform, right on your laptop



PROBLEM



SOLVING “DEPLOY AND PRAY”

Netlify Dev: Wrapping the DevServer








New Commands, New Config

Branch: master ▾ netlify-dev-plugin / src / commands / dev /

 biilmann	Fix live tunnel port when proxyPort is taken	...
..		
 exec.js	make tests actually work	
 index.js	Fix live tunnel port when proxyPort is taken	

```
# netlify.toml dev block example with _redirect file
[dev]
  command="yarn dev"
  port=5000
  publish="public" # this is new
```

Branch: master ▾ netlify-dev-plugin / src / commands / functions /

 sw-yx	fix broken templating
..	
 build.js	make tests actually work
 create.js	fix broken templating
 index.js	unhide functions command for docs
 list.js	make tests actually work

```
create-react-app-lambda ➤ master ➤ nfc
? Pick a template js-fauna-graphql
? name your function: fauna-graphql
◆ Creating function fauna-graphql
◆ Created built-lambda/fauna-graphql/fauna-graphql.js
◆ Created built-lambda/fauna-graphql/package.json
◆ Created built-lambda/fauna-graphql/schema.graphql
◆ Created built-lambda/fauna-graphql/sync-schema.js
✓ installed dependencies for fauna-graphql
◆ checking Netlify APIs...
◆ installing addon: fauna
Creating addon
Add-on "fauna" created for netlify-gotrue-in-react
◆ Injected addon env var: FAUNADB_ADMIN_SECRET
◆ Injected addon env var: FAUNADB_SERVER_SECRET
◆ Injected addon env var: FAUNADB_CLIENT_SECRET
? This template has an optional setup script that runs after addon install. This can be helpful
for first time users to try out templates. Run the script? (y/N) █
```

Private Dev with Oclif Plugins

netlify-dev-plugin

Netlify CLI plugin for local dev experience. [If you're interested to work on Netlify Dev and other product initiatives fulltime, we are hiring.](#)

Contributing/Local Development

Thanks for contributing! You'll need to follow these steps to run Netlify CLI and `netlify-dev-plugin` locally:

1. uninstall any globally installed versions of `netlify-cli`
2. clone and install deps for [netlify/cli](#)
3. `npm link` from inside the `cli` folder
4. clone and install deps for this repo
5. inside the `netlify-dev-plugin` folder, run `yarn link`
6. inside the `cli` folder, run `yarn link "netlify-dev-plugin"`

Now you're both ready to start testing `netlify dev` and to contribute to the project! Note these are untested instructions, please get in touch if you're unable to follow them clearly and we'll work with you. Or ping [@swyx](#).

 **Hacker News** new | threads | past | comments | ask

▲ Netlify Dev (netlify.com)

873 points by cift 51 days ago | hide | past | web | favorite | 229 comments | i

Netlify **Dev** Requirements

- Read **netlify.toml** config
- Check **login** state
- Check **site link** state
- Check **functions** folder exists
- Check **_redirects** folder exists
- Respect **flag** overrides
- **Prompt** for what's missing

Discovering 12 Factor Apps

- Help Docs
- Flags > Args
- --version
- stdout vs. stderr
- Errors, DEBUG=*
- Be **FANCY**
- Prompting
- Tables
- (Perceived) Speed
- Contributions
- Sub:commands
- XDG-spec

CLI Cheatsheet

<https://github.com/sw-yx/cli-cheatsheet>

PART II

THE 13th FACTOR



@swyx



BABEL

Adaptive Web Design

Crafting Rich Experiences with Progressive Enhancement

By Aaron Gustafson

Adaptive user interface

From Wikipedia, the free encyclopedia

For other uses, see [AUI \(disambiguation\)](#).

An **adaptive user interface** (also known as **AUI**) is a [user interface](#) (UI) which adapts, that is changes, its layout and elements to the needs of the user or context and is similarly alterable by each user.^{[1][2]}

These mutually reciprocal qualities of both adapting and being adaptable are, in a true AUI, also innate to elements that comprise the interface's components; portions of the interface might adapt to and affect other portions of the interface.

This later mechanism is usually employed to integrate two logically distinct components, such as an interactive document and an [application](#) (e.g. a [web browser](#)) into one seamless whole.

The user adaptation is often a negotiated process, as an adaptive user interface's designers ignore where user interface components ought to go while affording a means by which both the designers and the user can determine their placement, often (though not always) in a semi-automated, if not fully automated manner.

An AUI is primarily created based on the features of the system, and the knowledge levels of the users that will utilize it.

Contents [\[hide\]](#)

1 Advantages

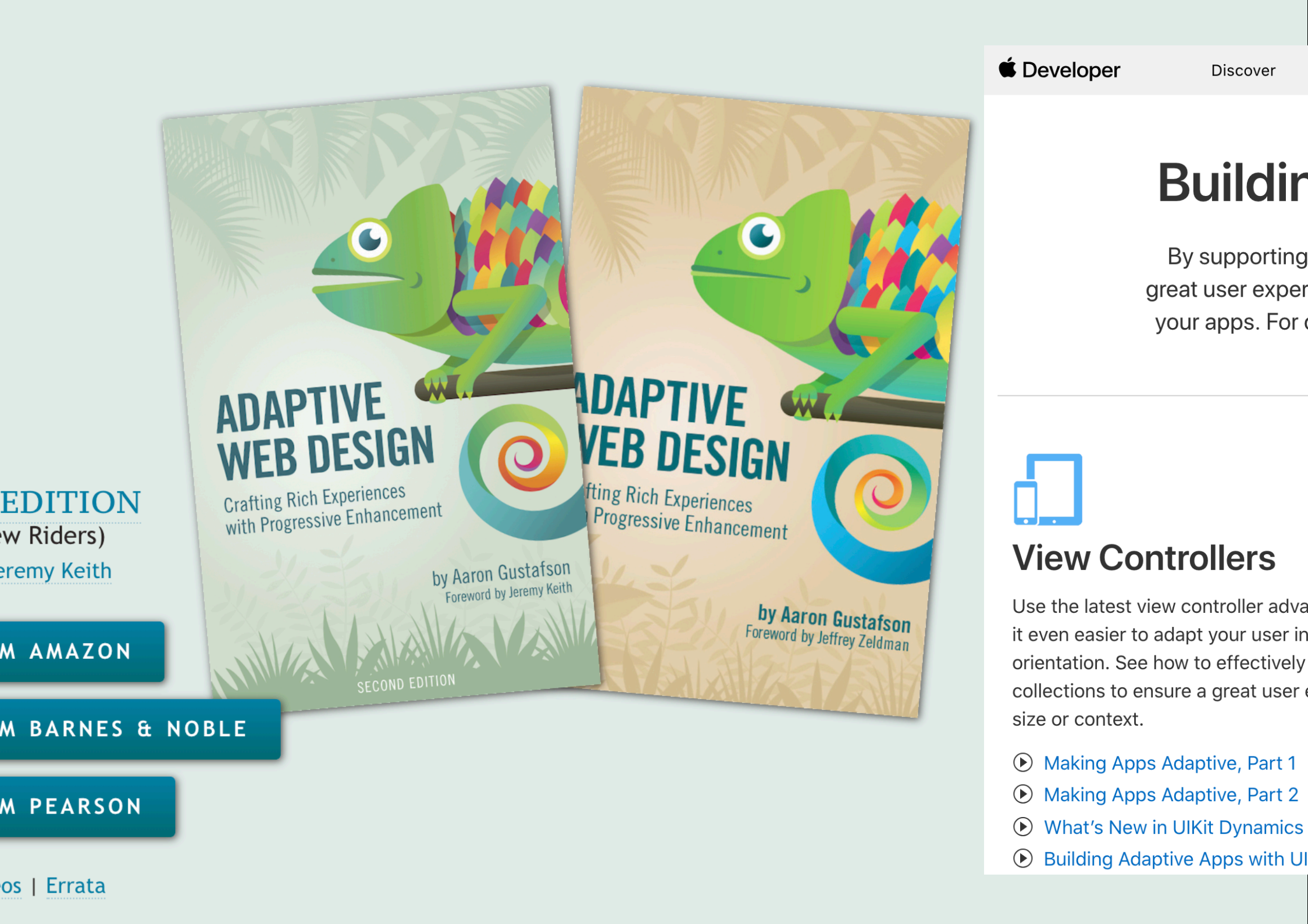
2 Disadvantages

3 Types

3.1 Adaptive presentation

3.2 Adaptive navigation

4 Uses in industry



Apple Developer

Discover

Design

Develop


Distribute

Support

Account

Building Adaptive User Interfaces

By supporting displays of any size and orientation, your iOS apps can deliver great user experiences. Use these resources to create adaptive user interfaces in your apps. For design recommendations, read the [Human Interface Guidelines](#).



View Controllers


Use the latest view controller advancements in UIKit to make it even easier to adapt your user interface to any size or orientation. See how to effectively use size classes and trait collections to ensure a great user experience for any display size or context.

▶ Making Apps Adaptive, Part 1

▶ Making Apps Adaptive, Part 2

▶ What's New in UIKit Dynamics and Visual Effects

▶ Building Adaptive Apps with UIKit



Dynamic Text

TextKit is the powerful text engine and API in iOS that provides sophisticated text handling and typesetting capabilities. Learn how to use TextKit to draw and manage text with adaptive user interfaces.

📄 [Text Programming Guide for iOS](#)

▶ [Introducing Text Kit](#)

▶ [Advanced Text Layouts and Effects with Text Kit](#)

▶ [Using Fonts with Text Kit](#)

Adaptive Interfaces

“Instead of designing web pages and content to respond to various devices... developers are trying to create interfaces that respond to individual users. These interfaces would **adapt on the fly to the current user and **collect data over time** to anticipate each user’s actions and preferences.”**

<https://speckyboy.com/adaptive-user-interfaces/>

THE 13th FACTOR STATE

PROBLEM I

STATE IS HARD

```
fs.writeFile
```

```
fs.writeFile
Object.as
```

```
)
```

```
ify(data));
```

```
ify(
```

write-file-atomic

This is an extension for node's `fs.writeFile` that makes its operation atomic and allows you set ownership (uid/gid of the file).

```
var writeFileAtomic = require('write-file-atomic')
writeFileAtomic(filename, data, [options], [callback])
```

- filename **String**
- data **String** | **Buffer**
- options **Object** | **String**
 - chown **Object** default, uid & gid of existing file, if any
 - uid **Number**
 - gid **Number**
 - encoding **String** | **Null** default = 'utf8'
 - fsync **Boolean** default = true
 - mode **Number** default, from existing file, if any
 - tmpfileCreated **Function** called when the tmpfile is created
- callback **Function**

Atomically and asynchronously writes data to a file, replacing the file if it already exists. data can be a string or a buffer.

```
path.resolve(process.cwd(), 'foo', '/store.json')
```

```
path.join(process.cwd(), 'foo', '/store.json')
```

```
path.resolve(process.cwd(), 'store.json')
```

```
path.join(process.cwd(), 'store.json')
```

```
path.resolve(__dirname, 'store.json')
```

```
path.join(__dirname, 'store.json')
```

Deploying in windows creates paths with backslashes that don't work once deployed #84

Edit

New issue

Closed

futuregerald opened this issue on Feb 20, 2018 · 3 comments



Add a library for cross-OS (okay, Windows) compatibility :) #45

Closed in #63

verythorough opened this issue on Mar 27 · 1 comment

Edit

New issue



veryth

The ne
with thIn Slac
anotheRelater
env, if

In any

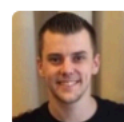
1

Respect the XDG base directory specification #152

Open

dsifford opened this issue on Jul 4, 2018 · 8 comments

New issue



dsifford on Jul 4, 2018 • edited ▾

+ 😊 ...

Hello,

I searched in the issues here and shockingly I haven't found this mentioned.

Currently, cosmiconfig searches up the parent tree from the working directory, stopping at `$HOME`, which is fine and most, I would assume, are okay with this.

However, for filesystem psychopaths like myself, it pains me to have to move any sort of user configuration file out `$XDG_CONFIG_HOME/project-name/` directory (usually, `~/.config/project-name/`) and pollute my home directory.

Any thoughts on adding this as a final search path?

You can read up on the specification [here](#) and [here](#).

In short, using prettier as an example, I'd imagine the workflow to go like this...

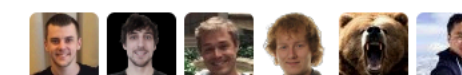
1. Perform the current lookup traversal as it is done right now.
2. If nothing found, check the following:
 - `$XDG_CONFIG_HOME/prettier/config`

🔊 Unsubscribe



Mark as unread

6 participants



Don't roll your own

- <https://github.com/sindresorhus/conf>
- <https://github.com/jonschlinkert/data-store>
- <https://github.com/davidtheclark/cosmicconfig>
(useful for reading config in .rc files as well but needs config for XDG compliance)

```
const Conf = require('conf');
```

```
const config = new Conf();
```

```
config.set('unicorn', '🦄');  
console.log(config.get('unicorn'));  
//=> '🦄'
```

```
// Use dot-notation to access nested properties  
config.set('foo.bar', true);  
console.log(config.get('foo'));  
//=> {bar: true}
```

```
config.delete('unicorn');  
console.log(config.get('unicorn'));  
//=> undefined
```

A full model of CLI State

- CLI Flags
- Project State
- Project Filesystem
- Machine State
- **Remote** user account settings
- **Remote** team account settings
- **Remote** global defaults

Solution I: *cli-state*

```
import {  
  initCLIState,  
  globalState,  
  projectState  
} from 'cli-state';
```

Offline State = Cache?

- CLI Flags
- Project State
- Project Filesystem
- Machine State
- **Cached** user account settings
- **Cached** team account settings
- **Cached** global defaults

PROBLEM II

CLI'S AS INTOLERANT PROCEDURE CALLS

```
docs-spike — -bash — 72x21
[todds-mbp-2:docs-spike toddmorey$ netlify dev
Netlify Dev ♦ Starting Netlify Dev...
Netlify Dev ♦ No dev server detected, using simple static server
Netlify Dev ♦ Unable to determine public folder for the dev server.
Setup a netlify.toml file with a [dev] section to specify your dev server settings.
[todds-mbp-2:docs-spike toddmorey$ netlify dev
Netlify Dev ♦ Starting Netlify Dev...
Netlify Dev ♦ Overriding dist with setting derived from netlify.toml [dev] block: null
Netlify Dev ♦ No dev server detected, using simple static server
Netlify Dev ♦ Unable to determine public folder for the dev server.
Setup a netlify.toml file with a [dev] section to specify your dev server settings.
```

▲ everdev 51 days ago [-]

Looks super cool and super happy with Netlify hosting.

I just gave it a spin though and got this:

```
Netlify Dev ♦ Starting Netlify Dev...
Netlify Dev ♦ Overriding dist with setting derived from netlify.toml [dev] block: null
Netlify Dev ♦ No dev server detected, using simple static server
Netlify Dev ♦ Unable to determine public folder for the dev server.
Setup a netlify.toml file with a [dev] section to specify your dev server settings.
```

But the blog post and the TOML reference (<https://www.netlify.com/docs/netlify-toml-reference/>) don't seem to include details on what to include to work.

Anyone have this tool working locally?

* 7 points by swyx 51 days ago [-]

hello! thanks for the report! i just pushed a small patch that now has some more helpful messages.. havent ironed out all the states yet and cant keep everything in my head!

<https://github.com/netlify/netlify-dev-plugin/commit/1c6df00...>

Context is hard

↕ You Retweeted



Jen Gentleman 🌸
@JenMsft

>> What do we want?

Natural language processing!

>> When do we want it?

When do we want what?

11:24 AM · May 25, 2019 · [Twitter Web App](#)

2K Retweets **9.6K** Likes

Handling Incomplete State

- **Worst: Silent Exit**
- **OK: Throw Error**
- **Better: Error Message**

5. Handle things going wrong

Things go wrong in CLIs much more often than in web apps. Without a UI to guide the user, the only thing we can do is display an error to the user. This is expected behavior and part of using any CLI.

First and foremost, make your errors informative. A great error message should contain the following:

1. Error code
2. Error title
3. Error description (Optional)
4. How to fix the error
5. URL for more information

For example, if our CLI errored out with a file permission issue, we could show the following:

```
$ myapp dump -o myfile.out
Error: EPERM - Invalid permissions on myfile.out
Cannot write to myfile.out, file does not have write permissions.
Fix with: chmod +w myfile.out
https://github.com/jdxcode/myapp
```

Handling Incomplete State

- **Worst: Silent Exit**
- **OK: Throw Error**
- **Better: Error Message**
- **Good: Prompt for fix**
- **Great: Adaptive prompting**

```
class Example2 extends Command {  
  // ...  
  async run() {  
    // ...  
    myBusinessLogic(state.name)  
    // throws!  
  }  
}
```

```
class Example2 extends Command {  
  // ...  
  async run() {  
    // ...  
    if (state.name) {  
      myBusinessLogic(state.name)  
    }  
    // silent error  
  }  
}
```

```
const chalk = require("chalk")
class Example2 extends Command {
  // ...
  async run() {
    // ...
    if (state.name) {
      myBusinessLogic(state.name)
    } else {
      this.error(`you did not provide ${chalk.yellow(name)}. please retry`)
    }
  }
}
```

```
const chalk = require("chalk")
class Example2 extends Command {
  // ...
  async run() {
    // ...
    if (state.name) {
      myBusinessLogic(state.name)
    } else {
      this.error(`Error code`)
      this.error(`Error title`)
      this.error(`Error description (Optional)`)
      this.error(`How to fix the error URL for more information`)
    }
  }
}
```

```
const chalk = require("chalk")
const { prompt } = require("enquirer")
class Example2 extends Command {
  // ...
  async run() {
    // ...
    if (state.name) {
      myBusinessLogic(state.name)
    } else {
      const name = prompt("give me a name")
      if (name) {
        myBusinessLogic(name)
      } else {
        this.error(`Error code`)
        this.error(`Error title`)
        this.error(`Error description (Optional)`)
        this.error(`How to fix the error URL for more information`)
      }
    }
  }
}
```

```
const chalk = require("chalk")
const { prompt } = require("enquirer")
class Example2 extends Command {
  // ...
  async run() {
    // ...
    if (state.name) {
      myBusinessLogic(state.name)
    } else {
      const name = prompt("give me a name")
      if (name) {
        console.log("next time you can pass a --name flag!")
        myBusinessLogic(name)
      } else {
        this.error(`Error code`)
        this.error(`Error title`)
        this.error(`Error description (Optional)`)
        this.error(`How to fix the error URL for more information`)
      }
    }
  }
}
```

KEY REALIZATION

WE WRITE CLI'S

LIKE WE WROTE JQUERY

mycli login  login.js

mycli deploy  deploy.js



Getting Started

[Introduction](#)[Features](#)[FAQs](#)[Single-command CLI](#)[Multi-command CLI](#)[Generator Commands](#)

API Reference

[Commands](#)[Command Arguments](#)[Command Flags](#)[Configuration](#)[Topics](#)[Hooks](#)[Plugins](#)

How to

[Release](#)[Testing](#)[Running Commands](#)[Programmatically](#)[Aliases](#)[Custom Error Objects](#)

Running Commands Programmatically

[EDIT](#)

If you need to run a command from another, or programmatically run a command in another codebase, there are a couple options.

First, it is **generally a bad idea to run a command directly** as the command exports a user interface, not a code interface. It's a design smell that should rarely (if ever) be used. Generally speaking, it's better to break up the code so that it can be called directly rather than as a command. We'll show this better method first.

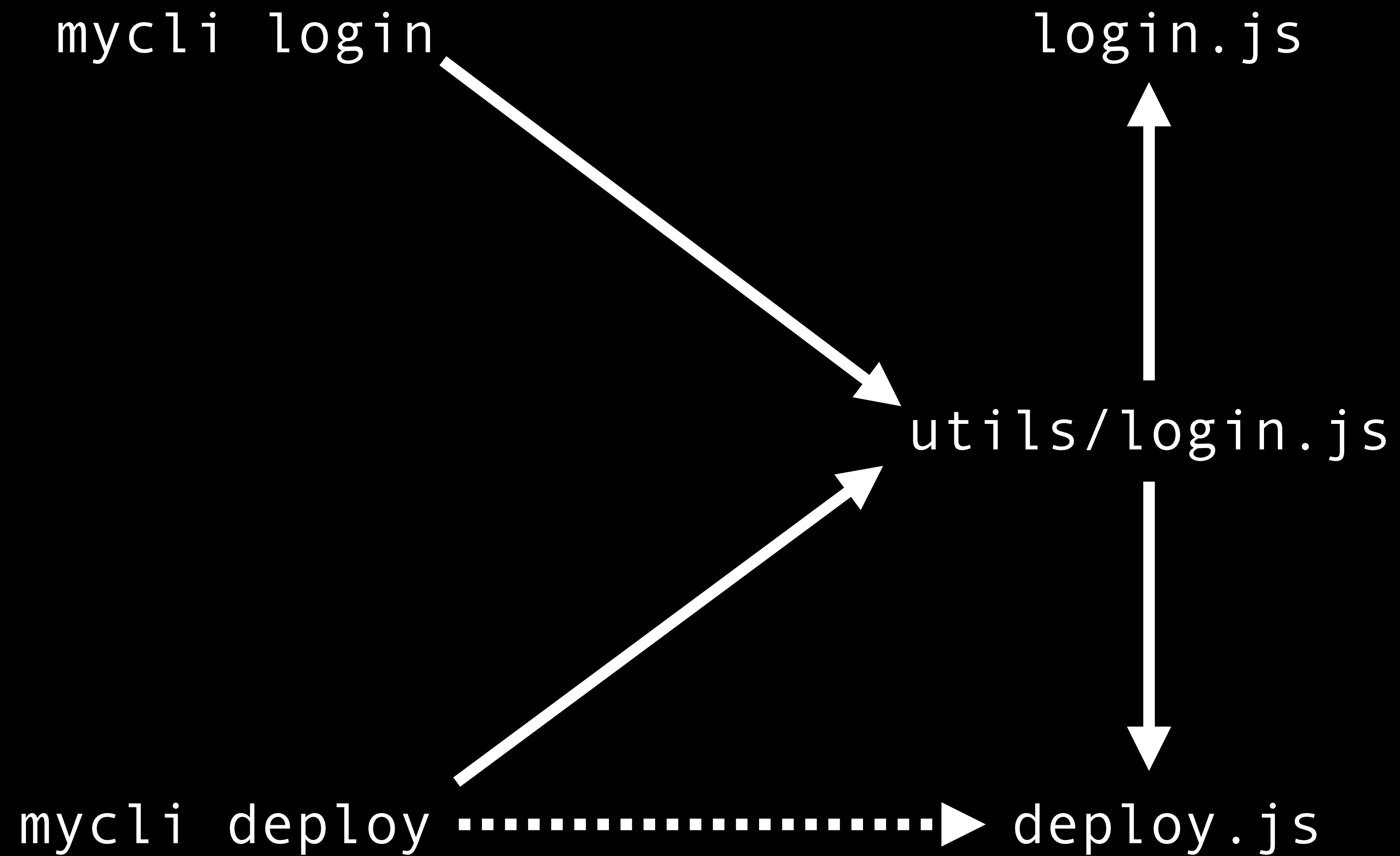
Sharing code with modules

For example, if we use `heroku config` as an example, we could have a command that outputs the config vars of an app to the screen like this:

`./src/commands/config.ts`

```
export class HerokuConfig extends Command {
  static flags = {
    app: flags.string({required: true})
  }

  async run() {
    const {flags} = this.parse(HerokuConfig)
    const config = await api.get(`/apps/${flags.app}/config-vars`)
    for (let [key, value] of Object.entries(config)) {
      this.log(`${key}=${value}`)
    }
  }
}
```



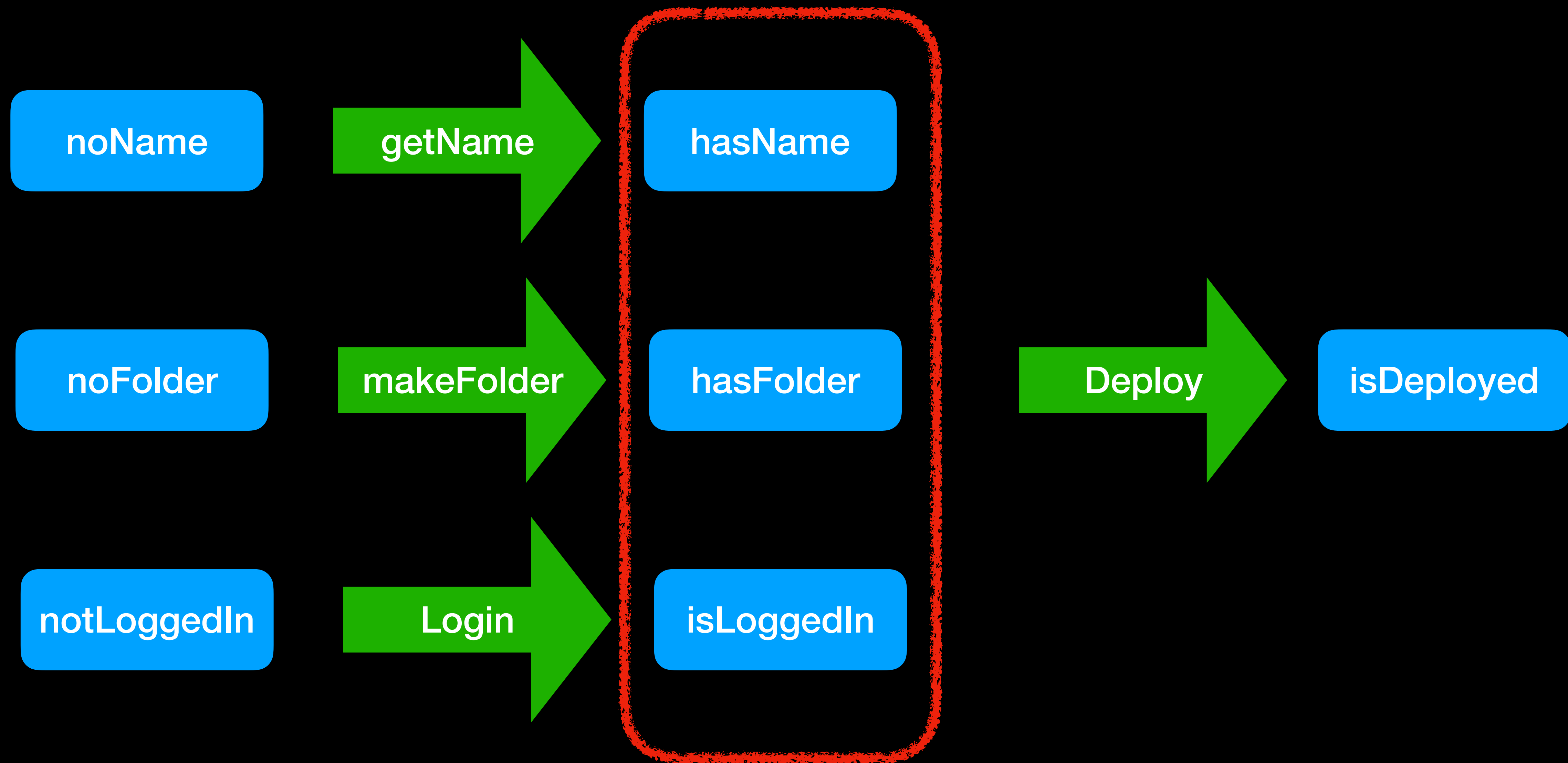
KEY REALIZATION

COMMAND \neq INTENT

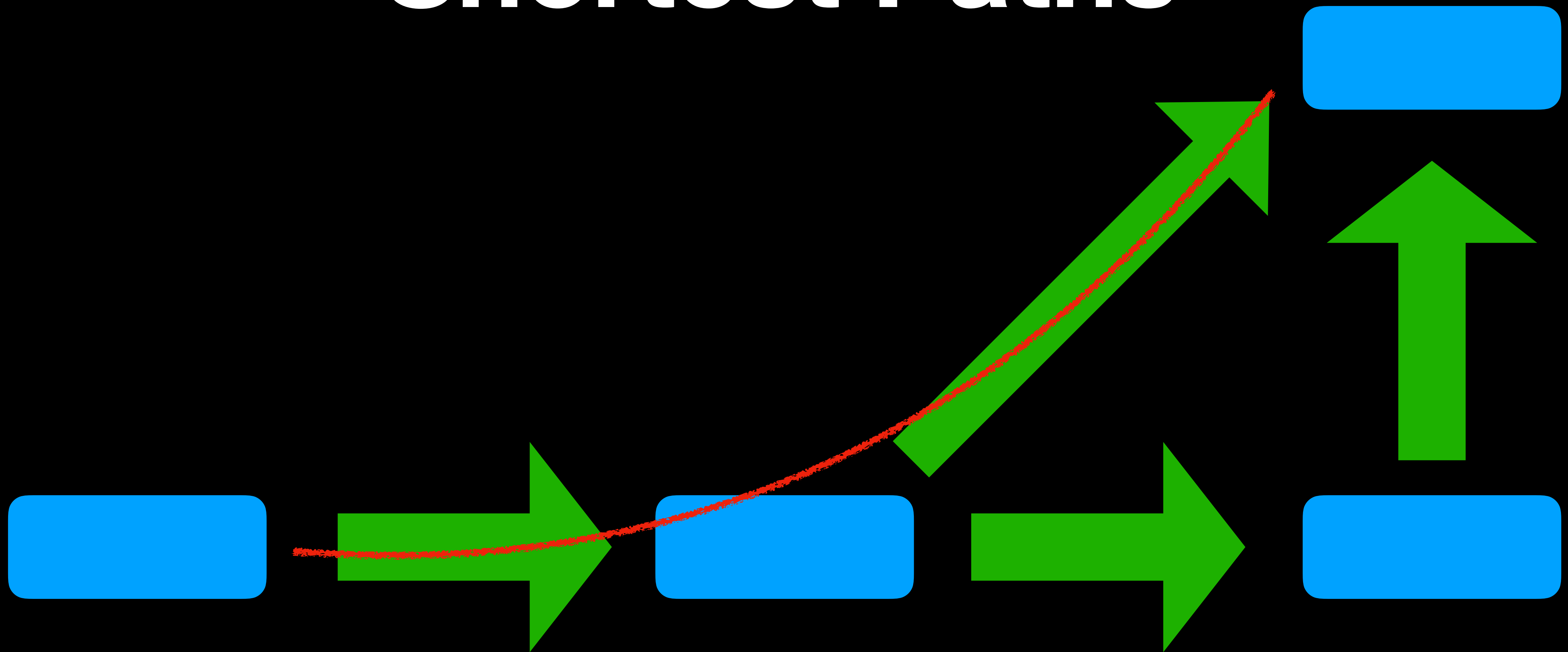
Use a State Machine

<https://statecharts.github.io/xstate-viz/>





Shortest Paths



<https://xstate.js.org/docs/packages/xstate-graph/#api>

Solution II: *cli-state-machine*

```
import {  
  Action,  
  State,  
  initStateMachine,  
  processStateMachine  
} from 'cli-state-machine'
```

Defining State

```
export const loggedInState: State = {  
  stateId: 'loggedIn',  
  getValue: async () => loginStatus,  
  assert: async (status: boolean) => status === true,  
}
```

Defining Action

```
export const loginAction: Action = {  
  actionId: 'loginAction',  
  beforeState: loggedOutState,  
  afterState: loggedInState,  
  execute: async () => {  
    // console.log('logging in')  
    loginStatus = true  
  },  
}
```

Defining another Action

```
export const deployAction: Action = {  
  actionId: 'deploy',  
  beforeState: loggedInState,  
  execute: async () => {  
    console.log('deployed!')  
  },  
}
```

Running the State Machine

```
initStateMachine( [  
  loginAction,  
  logoutAction,  
  deployAction  
])
```

```
// map intents to actions  
// let StateMachine search for valid adjacent states  
await processStateMachine(deployAction, {})
```

More To Do

- **Binding flags to States**
- **Helper functions for:**
 - **file/folder existence**
 - **Prompting for required field**
 - **???**
- **Visualization output**

**Principle: Componentized
and Declarative Business Logic**

the Full State Machine

```
// allActions and otherState defined above
initCLIState();
const cliState = { projectState, globalState, ...otherState }
initStateMachine(allActions)
await processStateMachine(deployAction, cliState)
```

Help us build it

netlify.com/careers

github.com/sw-yx/cli-state

github.com/sw-yx/cli-state-machine

ADAPTIVE *INTENT*-BASED *CLI STATE MACHINES*

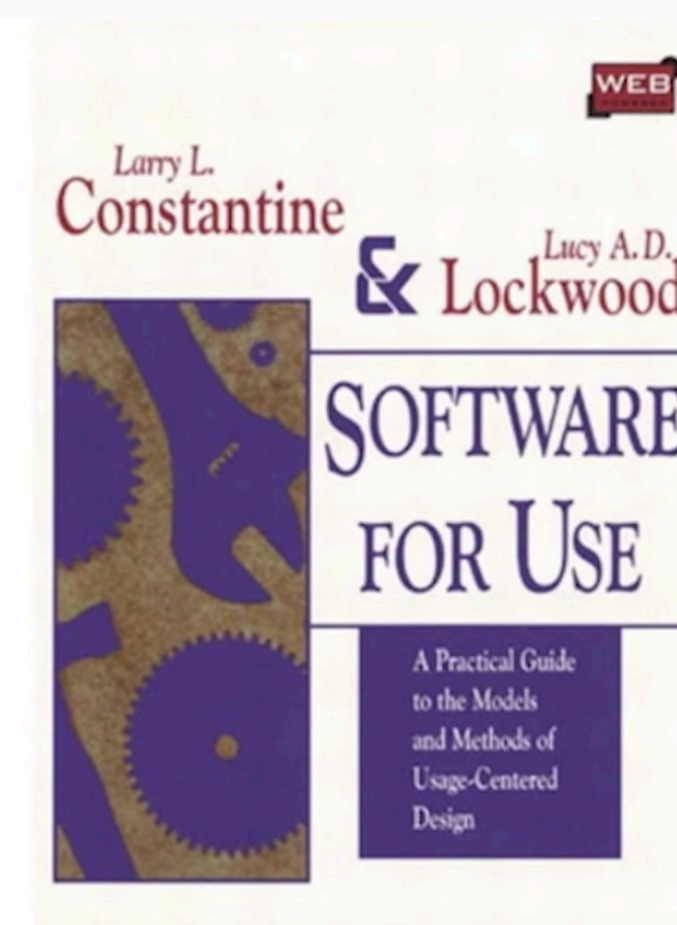
swyx.io/talks

npm i -g netlify-cli

Principles of HCI



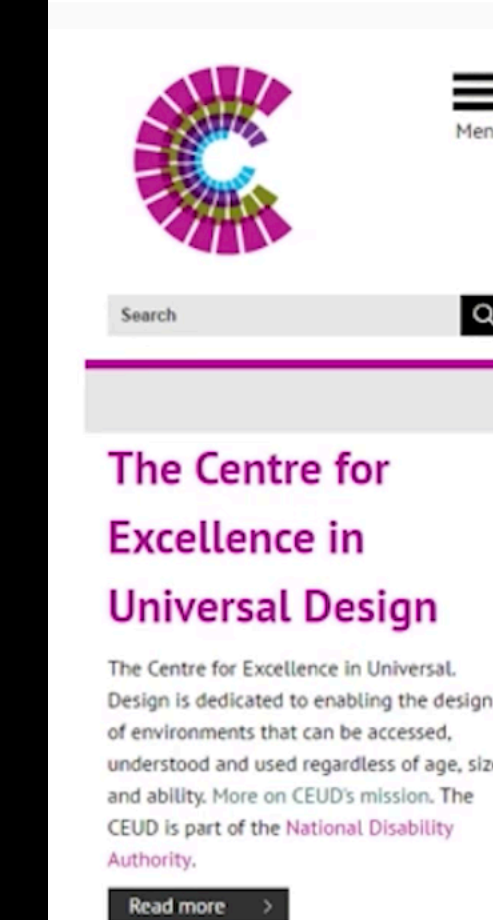
Don Norman's
Six Design
Principles



Larry
Constantine's
and Lucy
Lockwood's
Six Principles



Jakob Nielsen's
Ten Design
Heuristics



Ronald Mace's
Seven
Principles of
Universal
Design

12 General Principles of HCI



Discoverability



Feedback



Constraints



Mapping



Consistency



Affordances



Structure



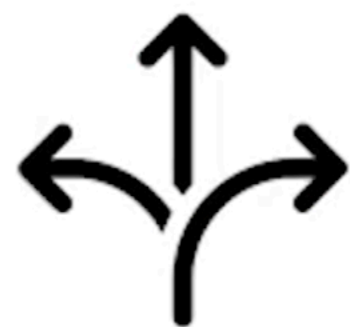
Simplicity



Tolerance



Equity



Flexibility



Perceptibility



Ease



Comfort



Documentation