Mastering CLI in TypeScript

by

Alex Korzhikov

&

Pavlik Kiselev

Amsterdam

16th of September 2019

Can you hear and see me well?

```
> Write to chat
> + good
> — problems with audio or video
```

Mastering CLI in TypeScript - Agenda

- CLI in Node
  - CLI Targets
  - Shell
  - Examples - npm, git
  - Basic Principles
- Hello World CLI in Node
  - `package.json`
  - `TypeScript`
- Hands-on CLI in Node
  - Tools Overview - prompt & Inquirer.js, `Commander.js`, `Vorpal`, `gluegun`
- Make it Work with `oclif`
  - Configure `oclif` project
  - Develop a command to slack hello world

Goals

- Understand Basic CLI Concepts
- Practice coding `JavaScript` & `TypeScript` CLI programs in `Node`
- Overview popular `npm` tools, libraries & frameworks for constructing CLIs
- Make an `oclif` CLI application to send Hello World notification to `slack`

To start with - Introduce yourself! 😀

- Who are you?
- What's your programming experience?
- Do you have questions about `CLI`, `Node` or `TypeScript`?

Who are we?

Alex Korzhikov

`JavaScript, Node, Web Components, TypeScript`

@ING @Otus

- Twitter: AlexKorzhikov
- Medium: korzio
- Github: korzio

Who are we?

Pavlik Kiselev

`JavaScript, Serverless, React, GraphQL`

@NonDutch

- LinkedIn
- Github: paulcodiny

Introduction

A command-line interface or command language interpreter (CLI), is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of lines of text (command lines). A program which handles the interface is called a command language interpreter or shell.

(c) Wiki

Porqué?

Which CLI program

- Do you like?
- Do you use the most?

- Why `CLI`?
- Why `JavaScript`?
- Why `Node`?
- Why `TypeScript`?

Why CLI?

`+`

- **Tools** for
  - improving **developer experience** and
  - task automation
- *which allow to gain even more!*
- *That's fun!*

`-` ?

CLI Targets

- CLI for API
- CLI for Domains
- CLI for Unification

Why JavaScript?

- JavaScript CLI for JavaScript Tasks
- JavaScript CLI for FrontEnd

```
npx cowsay hello cow
 _____________
< hello cow >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Why Node?

+

- Practice with `JavaScript`
- Atwood's Law - *any application that can be written in JavaScript, will eventually be written in JavaScript*
- Fast and easy to develop
- A rich infrastructure with all kinds of packages and libraries with `npm`
- Modules & plug'n'play

-?

- `Node` need to be installed?!

Why TypeScript?

\+

- Types for unifying protocols and interfaces, checked statically Ahead Of Time
  - According to To Type or Not to Type: Quantifying Detectable Bugs in JavaScript by Zheng Gao, Christian Bird, Christian Bird study, using TypeScript results in 15% decrease of bugs
  - Focus on API, not on implementation details
- OOP patterns and abstractions
- IDE help & support for writing code which will save your developers time

\-?

- Takes more time to develop and maintain projects

Shell

*a **program** that takes commands from the keyboard and gives them to the operating system to perform*

```
cat /etc/shells   # List of shells
cat /etc/passwd   # Default shell
```

What is your default shell?

- Interactive, non-interactive, login, non-login
- Built-in commands and scripts
- When a program is executed, a Bash process is forked

```
htop + bash
```

Principles Question

Which basic principles of designing a `CLI` program you might mention?

Examples

Principles

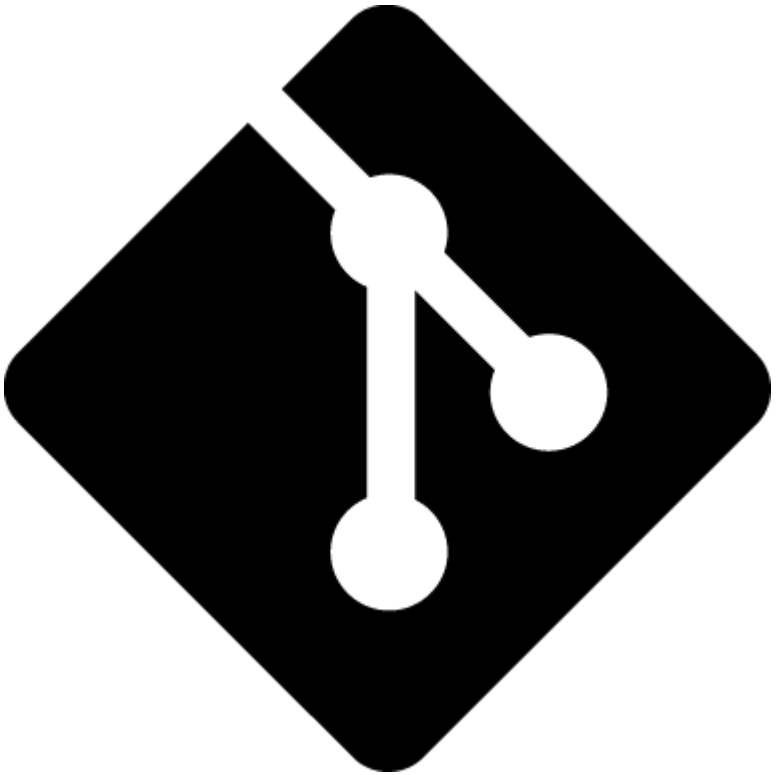
- Understand what's happening
  - `help`
  - `version`
  - `logs, messages, errors`
- `Do One Thing and Do It Well`

Top

- `git`
- `npm`

Generators & Developer Experience

- yeoman
- create-react-app
- angular-cli



Basic Principles

Q&A

-->

Practice

package.json

```
{
  "name": "my-hello-world-cli",
  "version": "1.0.0",
  "description": "Hello CLI",
  "main": "server.js",
  "bin": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "man" : "./man/doc.1"
}
```

- `main` - exports
- `bin` - make an executable `symlink` inside `PATH`, `./node_modules/.bin/`
- `url` - `npm bugs` - feedback on a package 🤗

Execution

- `shebang` specifies an interpeter in `*nix` systems

```
#!/usr/bin/env node
```

- `process.argv` contains arguments which a program is called with

What will be an output of running `server.js`?

```
console.log(process.argv)
```

```
node server hello world
```

- `repl` internal module provides a Read-Eval-Print-Loop (REPL) implementation

Practice - Hello World

Make the Hello World CLI in Node

```
mkdir my-hello-world-cli
npm init
echo "console.log('Hello CLI')" > index.js
npm start
npm install --global .
```

Parse arguments to show help message and version

```
my-hello-world-cli

Package description
Package version

Usage:
--help    Help documentation
--version Installed package version
```

Demo

Hello World CLI in Node

TypeScript

JavaScript that scales.
TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
Any browser. Any host. Any OS.
Open source.

- Anders Hejlsberg, 2012 @ Microsoft

Tools

- `typescript, tsc` - compile to `JavaScript`
- `tslint` - static code analysis, on a way to `eslint`
- `@types` - types definitions, @types/node
- `ts-node` - on-the-fly TypeScript execution for `Node`

Practice - Hello World with TypeScript

Make Hello World CLI with TypeScript

```
npm install -g typescript
tsc --init
mv index.js index.ts
npm install --save-dev @types/node
tsc
my-hello-world-cli
```

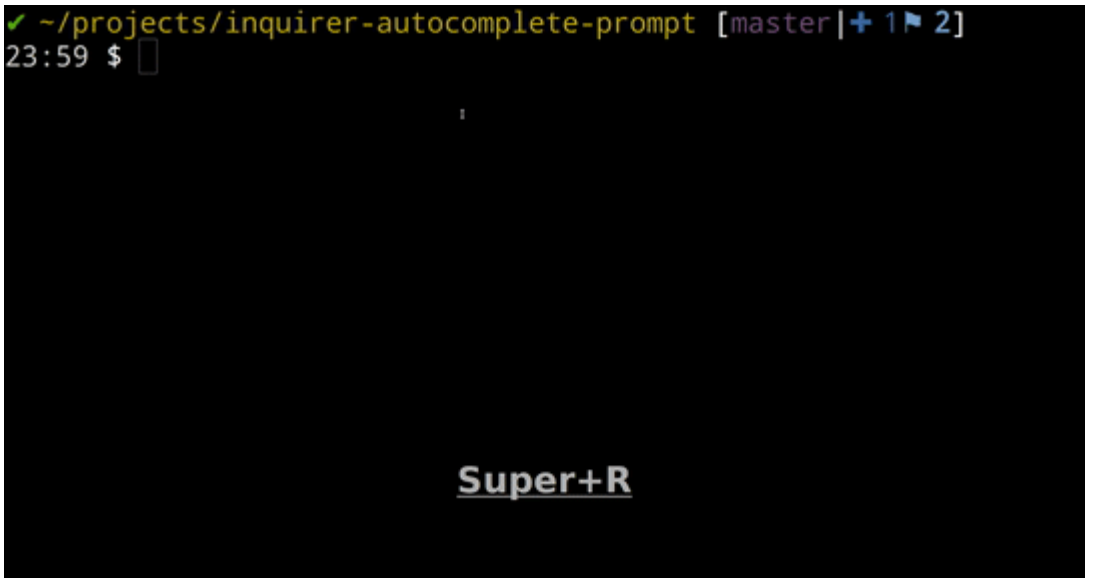
Commander.js

```
var program = require('commander')

program
  .version('0.1.0')
  .option('-p, --peppers', 'Add peppers')
  .option('-P, --pineapple', 'Add pineapple')
  .option('-b, --bbq-sauce', 'Add bbq sauce')
  .parse(process.argv)

console.log('you ordered a pizza with:')
if (program.peppers) console.log('  - peppers')
if (program.pineapple) console.log('  - pineapple')
if (program.bbqSauce) console.log('  - bbq')
console.log('  - %s cheese', program.cheese)
```

- Parse arguments
- Modular
- Auto-documentation

Inquirer

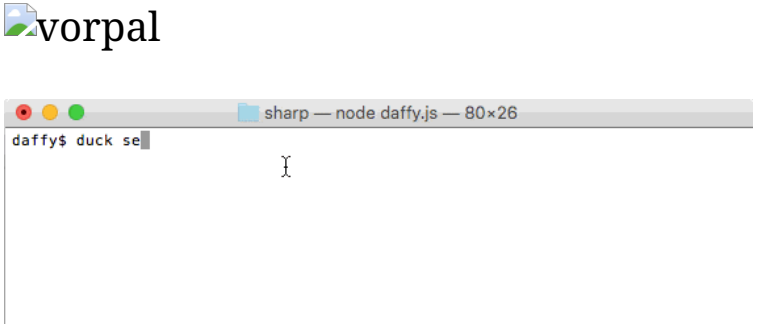
User's input



Alternatives

- `cli-ux`
- `prompt`

Vorpal



- Immersive Experience
- Auto documentation
- Parse arguments
- Input
- Autocompletion

Gluegun

toolkit for building Node-based command-line interfaces (CLIs) in TypeScript or modern JavaScript

```
module.exports = {
  name: 'generate', alias: ['g'],
  run: async (toolbox: GluegunToolbox) => {
    const {
      parameters: { first: project }, strings: { lowerCase, upperFirs
      template: { generate }, print: { info }, prompt: { ask }
    } = toolbox

    // ask a series of questions
    const { branch } = await ask([{ type: 'input', name: 'branch', me

    const fileName = 'CONTRIBUTING.md'
    const target = `${lowerCase(project)}/${fileName}`

    await generate({
      template: `${fileName}.ejs`,
      target, props: { project, branch },
    })

    info(`Generated ${fileName} file at ${target}`)
  },
}
```

Heroku, SalesForce framework to build CLIs

Features

- `TypeScript` (can be `JavaScript`)
- Auto-documentation
- Parse Arguments
  - Flags VS Arguments
- Code generation (with `yeoman`)
  - Single Commands VS Multi Commands
- Project' folders structure
- Hooks - a way to extend commands behavior
- Test & Build & Package

oclif Main Concepts

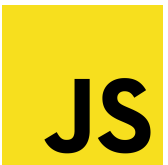
- Extend `Command` class

```
import Command from '@oclif/command'

export class MyCommand extends Command {
  static description = 'description of this example command'

  static flags = {
    help: flags.help({char: 'h'}),
    name: flags.string({char: 'n', description: 'name to print'}),
  }

  async run() {
    const { flags } = this.parse(MyCommand)

    console.log('running my command')
  }
}
```

Practice - Configure `oclif` project

Note - Project Management as CLI

Educational Open Source Project to practice with JavaScript, TypeScript, Node, oclif, Git, Web Components, and Project Management




```
npx oclif multi my-oclif-cli
cd my-oclif-cli
npm install -g .
my-oclif-cli hello
```

Practice - Make it Work

Make a command to send Hello World notification to `slack`

```
npm install @slack/webhook
npx oclif command slack
my-oclif-cli slack "Hello from @username"
```

```
const { IncomingWebhook } = require('@slack/webhook')
const url = process.env.SLACK_WEBHOOK_URL

const webhook = new IncomingWebhook(url)

// Send the notification
(async () => {
  await webhook.send({
    text: 'I\'ve got news for you...',
  })
})()
```

```
export SLACK_WEBHOOK_URL=https://hooks.slack.com/services/blabla
```

Feedback

Please share your feedback on Mastering CLI in TypeScript workshop

https://forms.gle/UZMRgpKLz2fuHBSe6

And one more

https://otus.pw/IpPd/

Demo

my-oclif-cli slack "Hello World!"

Libraries

- Decoration
  - `chalk` colors
  - `clui` output tables, status, charts
  - `progress` show status
  - `cli-table` print table data
  - `figlet` ASCII output
- Utilities
  - `clear` clear terminal
  - `cli-ux` oclif utilities for input output

Summary

- Understand Basic CLI Concepts
- Overviewed different `npm` packages for developing a CLI
- Practice with CLI in `Node` with `TypeScript` and popular frameworks & libraries

Docs

- Evolution of the Heroku CLI: 2008-2017
- 12 Factor CLI Apps - Heroku
- Building Great CLI Experiences in Node - Jeff Dickey, Heroku
- Build a JavaScript Command Line Interface (CLI) with Node.js — SitePoint

Thank you!

Questions?

Twitter: AlexKorzhikov
Medium: korzio
Github: korzio

LinkedIn
Github: paulcodiny