Well, yeah.

I’ve been creating CLI applications (the beardy admin says „scripts“) for a while now in various languages and settled in **Python** for the last years. Sadly however, in my company there seem to be only so many people, that develop in Python. So I had to leave Python behind and needed something else.

We’re a big e-commerce company creating shops for various customers using **Hybris** (and sometimes **Intershop**), so we have **a lot** of Java backend developers.

But Java… as a CLI-only application? While Java is good for a large software platform where many people develop simultaneously, it is hell for short one-task-scripts, that have to be ready in a few hours.

Besides the backend, we obviously also have a frontend team and, thus, many Javascript-developers. They’re basically browser-centric developers, but hey, what the hell. Should fit the slot.

So Javascript. Or let me correct that: **Ecmascript**. As you perhaps might know: Javascript has **nothing** to do with Java, so one can not simply project their Java-ideas onto Javascript. It’s a whole other language and methodology. You’ll have to adopt asynchronicity, callbacks, object-orienting by using prototypes and many other things.

Javascript outside the browser is run in Node.js (or io.js for that matter) nowadays. And while getting more comfortable with the whole Node infrastructure (say **Grunt**, **NPM**) I found, that the Node-people are doing various things suprisingly right.

I mean, yes, it’s a new rather new platform and there are various unstable things in it and Javascript is just… well, Javascript. But you can actually use the environment and language to create some pretty solid and testable code and all tools (various testing, coverage, mocking and api frameworks) exist for that.

Sure, when you’re coming from Python, where you have to invest a proper amount of work to make a script unreadable, you first have to adopt the various patterns that exist in the world of Javascript, but everything’s quite understandable and also easy to read (when developed right).

When I was using Python as a CLI language, I found **Cement** to be quite good. It’s a CLI-framework, that

- parses your arguments
- manages your configuration
- manages logging
- manages commands and subcommands (like „git clone …“)
- uses a template language for the output

„Wait, ‚uses a template language‘?“ Yes. And that’s quite an interesting fact, because why should modern CLI-scripts should be less „well-written“ than usual applications? Because they need to written in a short period of time? That’s no excuse! Why not invest a thought in a MVC-approach and separate the command line in- and output (view) from the logic (controller) and even an API or backend (model)? Once you think of it, it’s a quite cool feature. And while you’re at it, write unit and integration tests for your API and commands!

So what’s there in store for Javascript? There’s **seeli**, for example. Seeli is a CLI-framework, that

- parses your arguments
- manages commands and subcommands

That’s already quite a bit. What about the missing pieces?

- manages your configuration => **nconf**
- manages logging => **winston**
- uses a template language for the output => **handlebars**

Put it together in a thoughtful way, some Grunt-magic and you get the same features in a nice, understandable architecture in Javascript.

If you’d like to try it: I’ve created a **grunt-init** template, that you can use to get these features (note: nconf isn’t implemented in that template yet). Download it from **its repository** and put it in ~/.grunt-init. Afterwards just run

```
grunt-init grunt-init-seeli
```

in an empty directory. Check out the **Readme** for more information.