Clojure and Unix

James A. Overton

Clojure Toronto Meetup, 2015-08-18

Clojure and Unix

Learning Clojure

- learning Clojure can be hard
- I was used to imperative programming: Perl, Python, JavaScript, Java
- my code was full of `for` loops and `if` statements

Data Flow

- immutable data and the ->> macro changed my life
- I had to stop thinking of my code as poking at the data
- I started thinking of data flowing through code

Tables and Batches

- most of my Clojure code does batch processing
- source data usually comes from tables: CSV, SQL, Excel
- I convert rows to maps, tables to lists of maps
- I define little functions for updating those maps
- I work with lists of maps using `filter` and `map`
- I use Prismatic's Graph library to build a dependency graphs of tasks

Annoyances

- I'm a functional programming convert
- Clojure is my favourite language
- but there are annoyances!
  - Clojure is too heavy for scripting tasks
  - the JVM is its own universe
  - getting Clojure adopted can be hard
- for a lot of stuff, I just can't use Clojure

The Old Ways

- the older I get, the more I appreciate old tools and code
  - Vim for everything
  - live in the terminal
  - write Lisp
- I've written enough crap using the "shiny new thing"
- I don't have time to reinvent the wheel

Mental Block

- when I was an imperative programmer ...
  - I loved Python: modern, clean, consistent syntax
  - I hated Bash: arcane symbols, imperative structures feel tacked on
- Bash didn't suit my programming mindset
- I didn't bother to learn more ...
- I only used the shell for one-liners

Change is Slow

- I started collecting my one-liners in Makefiles
- I started writing two-liners and three-liners
- I started adding more Unix tools to my toolkit
- I never had to write any `for` loops in Bash
- **things were great!**

A Second Look

- my Clojure code is
  - full of little functions
  - grouped into pipelines
  - processing lists of rows
  - connected in dependency graphs
- but I was using Unix to write
  - little tool executions
  - grouped into pipelines
  - processing lists of rows
  - connected in dependency graphs

Strangely Similar

Clojure	Unix
lists of maps	tables (tab separated lines)
`->` `->>`	pipes, `tee`
`filter`	`find`, `grep`
`map`, `apply`	call, `xargs`
conditions	`test`
strings, regex	`sed`, `awk`, `tr`
Prismatic Graph	`make`
`pmap`	`parallel`

List Manipulation

Clojure	Unix
`conj`, `concat`	`cat`
`take`, `drop`	`head`, `tail`
`sort`	`sort`
`count`	`wc`
`distinct`	`uniq`
`frequencies`	`uniq -c`
`range`	`seq`
`shuffle`	`shuf`

Other Stuff

Clojure	Unix
`slurp`	`curl`
`assoc`, `dissoc`	`cut`, `join`
`println`	`echo`
`str`, `format`	`paste`, `printf`

But, but, but!

- REPL: the shell is your REPL!
- obscure syntax:
  - you got used to parens!
  - many people are more comfortable with the shell
- mutability, concurrency:
  - write to a new location
  - pipes are efficient (like transducers!)
- Windows support: use Vagrant, VMs
- fancy algorithms: use Clojure for the tricky parts
- fancy data structures: use Clojure for that part
- servers: fine, use Clojure

Upshot

Clojure and Unix tools are both focused on processing sequences of lightweight data structures through composable pipelines.

Closing Thoughts

- the more deeply I understand how great Clojure is, the less I use it
- Unix tools are usually lighter and faster, and easier to integrate with other tools
- use Clojure for the hard parts!