Literate Programming with Org Mode

Josh Holbrook

August 2020

Outline

Who Am I?

- Hi I'm Josh
- I'm a data engineer by day
- I use Emacs as my primary code editor
- I use Org Mode to stay organized
- I've used Org Mode to write literate programs

What's Org Mode?

- Emacs has different modes for different buffers
- The `org-mode` package enables support for org files
- Org files are based on an older outline mode in Emacs
- Org is really good for implementing TODO lists, GTD and other "productivity systems"

Org Has a Lot of Features

- Outlines, sure
- TODO states, of course
- Capture templates
- Calendar management (agendas)

Features That Go Beyond Productivity

- Markdown-like text formatting
- Links, hypertext and otherwise
- Inline images (`C-c C-x C-v` to toggle rendering!)

Figure: My pet budgie, Korben

Features That Go WAY Beyond Productivity

- Full On Spreadsheets with Calc and Lisp Equations
- Content exports and publishing - like this presentation
- Code execution and literate programming with org-babel

Org is a Way of Life, Really

- Org is one of Emacs' killer features
- Org is often a gateway drug into Emacs
- Many (most) of the people in this meetup have probably used org mode in some capacity

Org-Babel? Code Execution?

- Org supports inline "blocks", including "source blocks"
- Org-babel adds slick features around these blocks
- For example, we can execute this block of Emacs lisp with `C-c C-c`
- We can edit it with `C-c '`

```
(message "hello world!")
```

The Origins of Literate Programming

- Literate programming was invented by Donald Knuth in the mid 80s
- Donald Knuth wanted to write computer programs that could be sensibly read by humans
- In other words, a literate program is also a human language essay (or presentation)
- The first implementation, called WEB, was oriented towards Pascal and TEX

How Does It Work?

- You write a document that has human language and code snippets interspersed
- You use a tool that can “tangle” the source code into something a computer can run
- That same tool can “weave” the source code into a pretty document

The State of the Art

- As you might imagine, WEB isn't really used anymore
- noweb was a highly influential tool for literate programming but is dilapidated and rarely used in 2020
- Haskell is one of the few languages with first-class support for literate programming
- Jupyter notebooks are sometimes referred to as literate - and there are literate frameworks such as nbdev - but aren't flexible enough to truly rise to the occasion

Then There's Org Mode

- Org mode has great support for literate programming
- This makes org mode unusual!

Let's Build a Node.js Web Server

- You don't have to know Node.js or JavaScript to understand what you're about to see
- We're going to go really fast, because we don't actually need/want to learn Node.js or Express today

Node apps use a tool called `npm` to manage projects, which read a file called `package.json` in the root of the project:

```
{
  "name": "hello-express",
  "version": "1.0.0",
  "description": "An example Express app",
  "author": "Josh Holbrook",
```

Our app will expose a server object in `./hello-express/index.js` and it'll run a file called `./hello-express/server.js` to actually start it:

```
"main": "index.js",
"scripts": {
  "start": "node ./server.js"
},
```

A Brief Sidebar on Licensing

We'll use the GPL of course:

```
"license": "GPL-3.0-or-later",
```

Our JavaScript Files Will Need License Headers

Using the "noweb" feature, we can write one license header and include it in all of our JavaScript files:

```
/* Copyright 2020 Josh Holbrook
 *
 * This file is part of Josh Holbrook's Literate Programming
 * for NYC Emacs.
 *
 * This presentation is free software: you can redistribute
 * under the terms of the GNU General Public License as pub
 * Software Foundation, either version 3 of the License, or
 * later version.
 *
 * This presentation is distributed in the hope that it wil
 * WITHOUT ANY WARRANTY; without even the implied warranty
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Pu
 * details.
 *
 * You should have received a copy of the GNU General Publi
```

We're Going to use Express

Express is a microframework for Node.js. We can add it to our dependencies inside our `package.json`:

```
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

A Brief Sidebar on npm Usage

Normally you modify your `package.json` using npm commands. To install Express given an existing `package.json`:

```
npm i express
```

First, we'll need to crack open our JavaScript files and add our licensing headers using the noweb feature:

```
/* Copyright 2020 Josh Holbrook
 *
 * This file is part of Josh Holbrook's Literate Programming
 * for NYC Emacs.
 *
 * This presentation is free software: you can redistribute
 * under the terms of the GNU General Public License as pub
 * Software Foundation, either version 3 of the License, or
 * later version.
 *
 * This presentation is distributed in the hope that it wil
 * WITHOUT ANY WARRANTY; without even the implied warranty
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Pu
 * details.
 *
 * You should have received a copy of the GNU General Publi
```

Now We'll Require Our Modules

Node.js uses a module system that predates JavaScript "es6 modules", based on a standard called CommonJS. Using it to pull in Express looks like this:

```
const express = require('express');
```

Our Server Will Return HTML

Normally one would use a "real" templating language and Express's views functionality, but today we'll use a function that uses a template string:

```
function render_message(message) {
  return `<html>
    <head>
      <title>${message}</title>
    </head>
    <body>
      <h1>${message}</h1>
    </body>
  </html>`;
}
```

Now We'll Create Our Express App And Route

```
const app = express();

app.get('/', (req, res) => {
```

We're Sending HTML So We Have To Set The Status and Header

```
res.status = 200;
res.header('content-type', 'text/html');
```

Now We Can Send The Response Data (And End The Response)

```
  res.end(render_message('HELLO EMACS NYC!'));
});
```

Don't Forget To Export!

This is another part of Node's module system.

```
module.exports = app;
```

To Run It, First Require The Core HTTP Module And Our App

```
const http = require('http');

const app = require('./index');
```

Then, Create a Server

```
const server = http.createServer(app);
```

Finally, Listen On Port 8080

```
server.listen(8080, () => {
  console.log('Listening on 8080...')
});
```

Now Let's Tangle It

```
C-c C-v C-t
```

Now We Can Run It

This will block Emacs, so don't run it with `C-c C-c`!

```
cd ./hello-express
npm i
npm start
```

Kill with ctrl-c in the terminal.

Once It's Running We Can Curl It

This you CAN run with `C-c C-c`:

```
curl localhost:8080
```

Now Let's Weave It

We can build this presentation using `C-c C-e`!

We Used Source Blocks Configured To Tangle To Files

```
#+BEGIN_SRC javascript :tangle ./hello-express/index.js
```

We Used The Noweb Feature To Inline License Files

```
#+NAME: license-header
#+BEGIN_SRC javascript

#+BEGIN_SRC javascript :tangle ./hello-express/index.js :now
<<license-header>>
```

We Included Multiple Languages

- JavaScript
- But also JSON

We Both Tangled and Weaved The Org File

- Tangle: `C-c C-v C-t`
- Weave: `C-c C-e`

My Emacs Setup

I have org files for my Emacs config and my Nextcloud instance

- The config file tangles Emacs lisp code to `~/.doom.d` (I use Doom)
- The ops file tangles Nix configs, Terraform files and Ansible playbooks for managing the cloud instance
- The ops file tangles both a Makefile and an Invoke-Build file for PowerShell/Windows
- The ops file is arranged by feature (not file) and includes notes on what I was trying to accomplish
- These could be in one file and are only split for access reasons (I use the Emacs config at work)

Leetcode problems

I have a folder in my private monorepo for hanging onto the source code for some challenging Leetcode problems I've encountered

- A program can include not just the code but the how/why - a full explanation of the solution
- A program can include alternate solutions to the same problem
- This one includes a LEET HACK for includes from other files

Cackledaemon

I wrote a project in PowerShell that runs the Emacs daemon in a tray icon

- Unlike a lot of Emacs projects it's using a .NET language
- It tangles into a PowerShell module, helper scripts and an Invoke-Build file
- Tests are included next to the code I want to test
- The program doubles as documentation of all the issues arising from running Emacs in Windows "natively" and is intended to be a reference as much as it is a framework
- The document exports into an abbreviated README
- It's open source! (GPLv3+)

Organizationally They're Quite Good

- Literate programs can be organized the way my brain is
- Multiple source types about related concepts can be kept next to each other
- Noweb features mean snippets can be defined in an appendix and inlined later

Literate Programs are Readable and Informative

- A literate program can double as the documentation of my goals and thought process
- Being able to run source blocks means I can also include directions on how to use everything

News Flash, Emacs is Good

- Using org means I can easily collapse and expand sections to navigate my programs
- Using Emacs means I can take advantage of all of my programming modes throughout

Editing Is Only As Good As Your Config

- When working with PowerShell, I found myself fighting the mode sometimes
- I also (if memory serves) don't have a runner for PowerShell so `C-c C-c` doesn't work for PowerShell blocks

Breaking A Code Snippet Into Multiple Blocks Can Confuse Your Editor

- I did this with the `package.json` file in the example
- I had to manually indent the code with the spacebar in places

Many Tools Don't Work

- You can't use npm to edit org source blocks directly
- You also can't run a beautifier or linter based on the tangled source
- Though, many Emacs modes (including the PowerShell mode) include autoformatting and linting features

You Can't Include Multiple Source Types In The Same File

- This would be handy for inlining code blocks inside of bash snippets
- This would work if org collected by filename and kept snippets in the order seen
- Org actually groups source by language type first and then for each block writes to the necessary file, putting the snippets out of order

Exporting and Tangling Don't Quite Work The Way I Want Them To

- Noweb includes are ran prior to exporting - womp womp
- Tangling targets can be specified for one file document-wide or a different document per block, but you can't specify some blocks but default the rest
- Exports can do includes based on headline but tangling can't - LEET HACK you can get around this by exporting to org and then tangling the export

Thanks!

- @jfhbrook on GitHub
- @jfhbrook on Twitter