Literate DevOps

FOSDEM 2016

Marc Hoffmann
fosdem2016@h-rd.org
Literate DevOps for Configuration Management

or

How to write, explain, document and run "infrastructure as code".
Literate DevOps

My Background:

- Scientific simulation models
- Configuration of simulation stack
- Reproducible models
- Automation
- Teaching and writing
- Company http://h-rd.org
About this Talk

Literate DevOps
Ideas by:
• Howard Abrams
• Donald Knuth
• Carsten Dominik
Mistakes by me :-}
Quick search, this talk #5
“Standard” DevOps

Two phases:

1. Bang head until server works
2. Capture effort into some automation tool like Puppet or Chef
What is Literate DevOps?

Literate DevOps is the combination of Literate Programming and DevOps:

"Documentation with embedded executable DevOps code"
What is Literate Programming?

• Documents with "commented out" code vs code with comments
• Invented by Donald Knuth in the 1980's
What is Literate Programming?

Main idea: Communication

- Ideas
- Concepts
- Code

for others (and yourself)
How it works

- Org Mode / Markdown File
- PDF / HTML
- Source Code
Tooling

- Example tooling based on emacs and org
- Developed by Howard Abrams
- https://github.com/howardabrams/
- Easy adaption to e.g. markdown
- Text/code based, so easy VC/git integration
Emacs and Org

- Emacs is THE extensible editor
- Org is an interactive writing/scripting/organizing/hyperlinking application for Emacs [http://orgmode.org](http://orgmode.org)
- Interactive code execution in org document by `Ctrl-C  Ctrl-C`
- Batch code and documentation export to html, markdown and code
- Great for DevOps idempotence: Apply and test recipes/manifests again and again with easy `Ctrl-C  Ctrl-C` for learning
Org Mode

- Great at thought collection and organization
- Hyperlinks for documents, code, remote execution
- System is build from small, simple parts
- Executable literate programming
- Mix (weave) code from multiple languages
Org Mode

Notes and resolution of work issues during the 'Fuzzy Bunny' sprint.

* Work Issues...
* Meeting Notes...
* Daily Scrum Status...
* Sprint Demonstration...
* Notes for Next Sprint...
# Sprint: Fuzzy Bunny

**Title:** Sprint: Fuzzy Bunny

**Author:** Howard Abrams

**Email:** howard.abrams@gmail.com

## Notes

and resolution of work issues during the 'Fuzzy Bunny' sprint.

### Work Issues...

** Extend Project Scope with Lint Checking | [[blah1][WC-152]]...**

** Create a Setup File for Better Installation | [[blah1][WC-134]]...**

** Verify the Installable Archive...**

** Set up a Chef Server for Deployment | [[blah1][WC-181]]...**

** Install Chef Binaries**

** Upload Cookbooks**

** Connect the Clients...**

** Deploy the Project Application | [[blah1][WC-182]]...**

** Install Python Server...**

** Install Apache with WSGI...**

** Create Local Dev Environment with Docker | [[blah1][WC-195]]...**

### Meeting Notes...

** Daily Scrum Status...**

** Sprint Demonstration...**

** Notes for Next Sprint...**
Org Mode

* Work Issues

** Extend Project Scope with Lint Checking

After researching many alternatives, including:
- [http://www.pylint.org](http://www.pylint.org) :: Individual's can customize errors and conventions.
- [http://pychecker.sourceforge.net](http://pychecker.sourceforge.net) :: hasn't been updated in years. Issue?
- [https://pypi.python.org/pypi/pep8](https://pypi.python.org/pypi/pep8) :: Guido's original style guide.
- [https://flake8.readthedocs.org/en/2.3.0/](https://flake8.readthedocs.org/en/2.3.0/) :: Integrate both =pep8= /and/ =pyflakes=.

Since it wraps pep8 as well as [https://pypi.python.org/pypi/pyflakes](https://pypi.python.org/pypi/pyflakes) error checking library, I'm sure that *Flake8* library should be sufficient for us.

Install it in a virtual environment with:

```
#BEGIN_SRC sh :exports code
  pynv virtualenv demo
  pynv activate demo
  pip install --upgrade flake8
#END_SRC
```

#+begin_example
Collecting virtualenv
  Using cached virtualenv-13.1.0-py2.py3-none-any.whl
Installing collected packages: virtualenv
Successfully installed virtualenv-13.1.0
Collecting flake8
  Using cached flake8-2.4.1-py2.py3-none-any.whl
Collecting pyflakes<0.9,>=0.8.1 (from flake8)
  Using cached pyflakes-0.8.1-py2.py3-none-any.whl
U:--- sprint-fuzzy-bunny.org Top (1,0) (Org Pabbrev Fill) 15:59 Mail
pabbrev scavenging...done
<table>
<thead>
<tr>
<th>Advanced Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote shell access through ssh</td>
</tr>
<tr>
<td>Remote file access through &quot;tramp&quot;</td>
</tr>
<tr>
<td>Remote sudo</td>
</tr>
</tbody>
</table>

/ssh:bastion.example.com|ssh:server.example.com/var/www/index.html

ssh access | (piped through bastion) to file on server
Cross Platform

- Emacs: Linux, Windows, OSX, *BSD, ARM, ...
- no X required, terminal
- non-interactive, command line exec
What is Literate Programming?

Code with comments:

```javascript
/* my comments */
var my_code = "abc";
/* my next comment */
```
What is Literate Programming?

Code with comments:

```plaintext
#+BEGIN_SRC
/* my comments */
var my_code = "abc";
/* my next comment */
#+END_SRC
```

Documentation with code:

```plaintext
#+BEGIN_SRC
my comments
/* var my_code = "abc"; */
my next comment
#+END_SRC
```
Literate DevOps

- Literate programming for DevOps: scripts/manifests/playbooks
- Term coined by Howard Abrams
- Provides structure and clarity for code
- Allows to communicate systems to others
Let's start with a simple example:

List the files in a target directory and count the words in the files.

```bash
#+NAME: lsfiles
#+BEGIN_SRC sh
  ls -lok *
#+END_SRC
```
Let's start with a simple example:

List the files in a target directory and count the words in the files.

```
#+NAME: lsfiles
#+BEGIN_SRC sh
   ls -lok *
#+END_SRC

#+RESULTS: lsfiles

| -rw-r--r-- |   1 | marc  |   0 | Jan | 27 | 15:07 | 1.txt |
| -rw-r--r-- |   1 | marc  |   0 | Jan | 27 | 15:07 | 2.txt |
| -rw-r--r-- |   1 | marc  |   0 | Jan | 27 | 15:07 | 3.txt |
| -rw-r--r-- |   1 | marc  | 319 | Jan | 27 | 15:11 | ex.org |
```
Let's start with a simple example:

List the files in a target directory and count the words in the files.

```
#+NAME: lsfiles
#+BEGIN_SRC sh
  ls -lok *
#+END_SRC

#+RESULTS: lsfiles
| -rw-r--r-- | 1  marc | 0  | Jan | 27 | 15:07 | 1.txt |
| -rw-r--r-- | 1  marc | 0  | Jan | 27 | 15:07 | 2.txt |
| -rw-r--r-- | 1  marc | 0  | Jan | 27 | 15:07 | 3.txt |
| -rw-r--r-- | 1  marc | 319| Jan | 27 | 15:11 | ex.org |
```

This blocks reads the `lsfiles` as a list, and just grabs the last column (the names) as the variable, `$FILES`:

```
#+BEGIN_SRC sh :var FILES=lsfiles[,7]
  wc -l $FILES
#+END_SRC
```
Let's start with a simple example:

List the files in a target directory and count the words in the files:

```sh
ls -lok *
```

This blocks reads the `lsfiles` as a list, and just grabs the last column (the names) as the variable, `=$FILES=`:

```sh
wc -l $FILES
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.txt</td>
<td>1.txt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2.txt</td>
<td>2.txt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3.txt</td>
<td>3.txt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ex.org</td>
<td>ex.org</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>total</td>
<td>total</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chef example from [[https://supermarket.chef.io/cookbooks/application]]:

```ruby
#+begin_src ruby
  application '/path/to/deploy' do
    owner 'root'
    group 'root'
  end
#+end_src
```

This example needs still to be refined to be used.
Chef example from [https://supermarket.chef.io/cookbooks/application]]:

```ruby
#+begin_src ruby
application '/path/to/deploy' do
  owner 'root'
  group 'root'
end
#+end_src
```

This example needs still to be refined to be used.

This snippet can be written to a file and remotely executed.
Literate DevOps

Chef example from [[https://supermarket.chef.io/cookbooks/application]]:

```
#+begin_src ruby
  application '/path/to/deploy' do
    owner 'root'
    group 'root'
  end
#+end_src
```

This example needs still to be refined to be used.
Mix Chef with Terraform and apex

Org allows you to mix and match multiple languages and stacks:

- Terraform (manages DNS, network, VM's)
- Chef/Puppet/Ansible
- apex (Go, “capistrano” for AWS Lambda)
Use Case: Terraform / Vagrant

- set up Vagrant
- put Terraform into Vagrant
- set up cloud / AWS
- provision with Puppet / Chef

complicated, mix of tools

use Literate DevOps
Use Case: apex / Vagrant

- set up Vagrant
- put apex into Vagrant
- set up AWS Lambda
- set up storage

complicated, mix of tools
use Literate DevOps
Use Cases

- both use cases have commonalities (Vagrant)
- both install something into Vagrant (Terraform, apex)
- both do some setup (puppet, AWS Lambda)
- with LitDevOps smooth transition between use cases

switch of tooling
use Literate DevOps
Re-use knowledge and code

Use Cases show:
- systems evolve
- tools change
- some things stay the same
- re-use
- no why and how documentation:

CHAOS
Infrastructure and tools change all the time.

New (and old) kids on the block:

- Google App Engine
- AWS Lambda
- serverless.com
- NIX, guix (no remote editing, docs still needed)
Literate DevOps

You can use Literate DevOps only during development. Export code once it's working.

(tooling not anymore required, beware! changes mix things up)
Literate DevOps

maybe: DocOps
Takeaway

Literate DevOps =

( infrastructure + documentation ) as code
Links

- https://en.wikipedia.org/wiki/Literate_programming
- http://h-rd.org/literate-devops
- http://orgmode.org
Literate DevOps

Slides at:
http://h-rd.org/literate-devops

Questions? I like to help, just ask the guy on the photo!

fosdem2016@h-rd.org